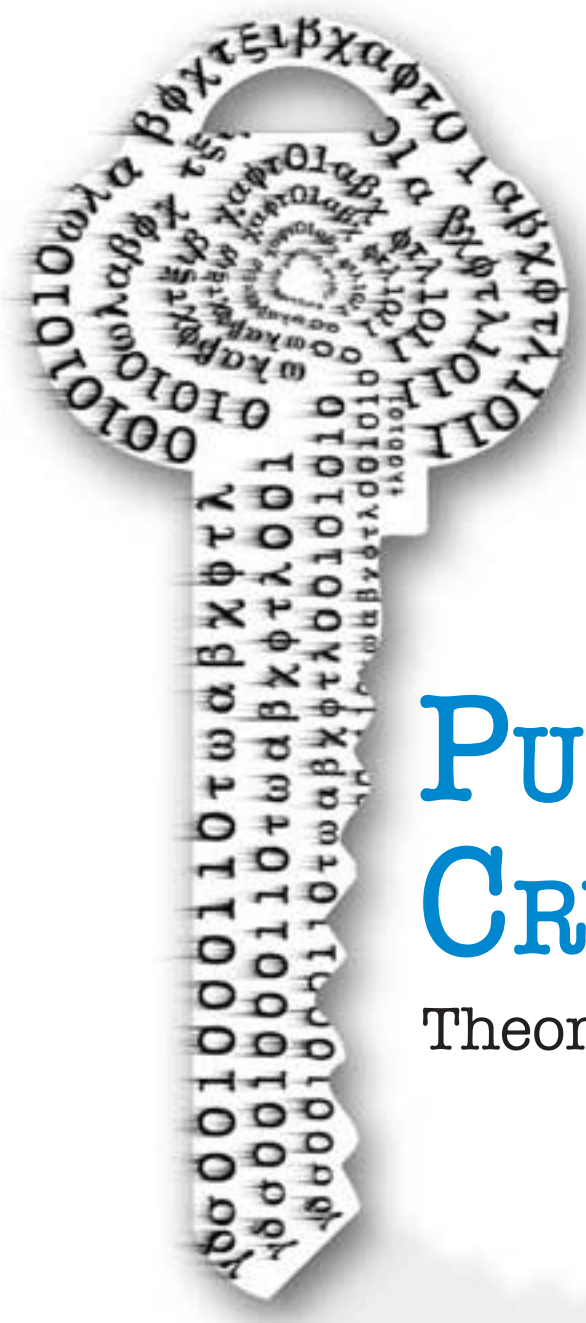


Abhijit Das
C. E. Veni Madhavan

Public-key Cryptography

Theory and Practice

Public-key Cryptography
Theory and Practice

ABHIJIT DAS
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur

C. E. VENI MADHAVAN
Department of Computer Science and Automation
Indian Institute of Science, Bangalore

Chennai • Delhi • Chandigarh
Upper Saddle River • Boston • London
Sydney • Singapore • Hong Kong • Toronto • Tokyo

Library of Congress Cataloging-in-Publication Data

Das, Abhijit & Madhavan, C E Veni
Public-keyCryptography / Abhijit Das & C E Veni Madhavan.
p. cm.
Includes bibliographical references and index.
ISBN-13: 978-8131700000 (pbk.)
1. Public-keyCryptography. I. Title.
HB172.5.D76 2009
339—dc22

2008010113

Copyright © 2009 Dorling Kindersley (India) Pvt. Ltd.
Licensees of Pearson Education in South Asia

This book is sold subject to the condition that it shall not, by way of trade or otherwise, be lent, resold, hired out, or otherwise circulated without the publisher's prior written consent in any form of binding or cover other than that in which it is published and without a similar condition including this condition being imposed on the subsequent purchaser and without limiting the rights under copyright reserved above, no part of this publication may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording or otherwise), without the prior written permission of both the copyright owner and the above-mentioned publisher of this book.

ISBN 978-81-317-0832-3

Head Office: 482 FIE, Patparganj, Delhi 110 092, India
Registered Office: 14 Local Shopping Centre, Panchsheel Park, New Delhi 110 017, India

Pearson Education Inc., Upper Saddle River, NJ
Pearson Education Ltd., London
Pearson Education Australia Pty, Limited, Sydney
Pearson Education Singapore, Pte. Ltd
Pearson Education North Asia Ltd, Hong Kong
Pearson Education Canada, Ltd., Toronto
Pearson Educacion de Mexico, S.A. de C.V.
Pearson Education-Japan, Tokyo
Pearson Education Malaysia, Pte. Ltd.

Contents

Preface **xiii**

Notations **xv**

1 Overview **1**
1.1 Introduction . . . 2
1.2 Common Cryptographic Primitives . . . 2
1.2.1 The Classical Problem: Secure Transmission of Messages . . . 2
Symmetric-key or secret-key cryptography . . . 4
Asymmetric-key or public-key cryptography . . . 4
1.2.2 Key Exchange . . . 5
1.2.3 Digital Signatures . . . 5
1.2.4 Entity Authentication . . . 6
1.2.5 Secret Sharing . . . 8
1.2.6 Hashing . . . 8
1.2.7 Certification . . . 9
1.3 Public-key Cryptography . . . 9
1.3.1 The Mathematical Problems . . . 9
1.3.2 Realization of Key Pairs . . . 10
1.3.3 Public-key Cryptanalysis . . . 11
1.4 Some Cryptographic Terms . . . 11
1.4.1 Models of Attacks . . . 12
1.4.2 Models of Passive Attacks . . . 12
1.4.3 Public Versus Private Algorithms . . . 13

2 Mathematical Concepts **15**
2.1 Introduction . . . 16
2.2 Sets, Relations and Functions . . . 16
2.2.1 Set Operations . . . 17
2.2.2 Relations . . . 17
2.2.3 Functions . . . 18
2.2.4 The Axioms of Mathematics . . . 19
Exercise Set 2.2 . . . 20
2.3 Groups . . . 21
2.3.1 Definition and Basic Properties . . . 21
2.3.2 Subgroups, Cosets and Quotient Groups . . . 23
2.3.3 Homomorphisms . . . 25
2.3.4 Generators and Orders . . . 26
2.3.5 Sylow's Theorem . . . 27
Exercise Set 2.3 . . . 29
2.4 Rings . . . 31
2.4.1 Definition and Basic Properties . . . 31
2.4.2 Subrings, Ideals and Quotient Rings . . . 34
2.4.3 Homomorphisms . . . 37
2.4.4 Factorization in Rings . . . 39
Exercise Set 2.4 . . . 42
2.5 Integers . . . 44

2.5.1 Divisibility . . . 44
2.5.2 Congruences . . . 45
2.5.3 Quadratic Residues . . . 48
2.5.4 Some Assorted Topics . . . 52
The prime number theorem . . . 53
Density of smooth integers . . . 54
The extended Riemann hypothesis . . . 54
Exercise Set 2.5 . . . 56
2.6 Polynomials . . . 57
2.6.1 Elementary Properties . . . 58
2.6.2 Roots of Polynomials . . . 59
2.6.3 Algebraic Elements and Extensions . . . 61
Exercise Set 2.6 . . . 63
2.7 Vector Spaces and Modules . . . 64
2.7.1 Vector Spaces . . . 65
2.7.2 Modules . . . 69
2.7.3 Algebras . . . 71
Exercise Set 2.7 . . . 72
2.8 Fields . . . 74
2.8.1 Properties of Field Extensions . . . 74
2.8.2 Splitting Fields and Algebraic Closure . . . 76
2.8.3 Elements of Galois Theory . . . 78
Exercise Set 2.8 . . . 79
2.9 Finite Fields . . . 80
2.9.1 Existence and Uniqueness of Finite Fields . . . 80
2.9.2 Polynomials over Finite Fields . . . 82
2.9.3 Representation of Finite Fields . . . 85
Exercise Set 2.9 . . . 88
2.10 Affine and Projective Curves . . . 90
2.10.1 Plane Curves . . . 90
2.10.2 Polynomial and Rational Functions on Plane Curves . . . 92
2.10.3 Maps Between Plane Curves . . . 95
2.10.4 Divisors on Plane Curves . . . 95
Exercise Set 2.10 . . . 97
2.11 Elliptic Curves . . . 98
2.11.1 The Weierstrass Equation . . . 98
2.11.2 The Elliptic Curve Group . . . 101
2.11.3 Elliptic Curves over Finite Fields . . . 106
Exercise Set 2.11 . . . 107
2.12 Hyperelliptic Curves . . . 111
2.12.1 The Defining Equations . . . 111
2.12.2 Polynomial and Rational Functions . . . 112
2.12.3 The Jacobian . . . 115
Exercise Set 2.12 . . . 118
2.13 Number Fields . . . 119
2.13.1 Some Commutative Algebra . . . 120
Ideal arithmetic . . . 120
Localization . . . 120
Integral dependence . . . 121

Noetherian rings . . . 123
Dedekind domains . . . 125
2.13.2 Number Fields and Rings . . . 125
2.13.3 Unique Factorization of Ideals . . . 131
2.13.4 Norms of Ideals . . . 135
2.13.5 Rational Primes in Number Rings . . . 137
2.13.6 Units in a Number Ring . . . 139
Exercise Set 2.13 . . . 139
2.14 p-adic Numbers . . . 143
2.14.1 The Arithmetic of p-adic Numbers . . . 143
2.14.2 The p-adic Valuation . . . 145
2.14.3 Hensel's Lemma . . . 149
Exercise Set 2.14 . . . 151
2.15 Statistical Methods . . . 154
2.15.1 Random Variables and Their Probability Distributions . . . 154
2.15.2 Operations on Random Variables . . . 155
2.15.3 Expectation, Variance and Correlation . . . 159
2.15.4 Some Famous Probability Distributions . . . 162
Uniform distribution . . . 162
Bernoulli distribution . . . 163
Normal distribution . . . 163
2.15.5 Sample Mean, Variation and Correlation . . . 164
Exercise Set 2.15 . . . 165

3 Algebraic and Number-theoretic Computations 173
3.1 Introduction . . . 174
3.2 Complexity Issues . . . 174
3.2.1 Order Notations . . . 175
3.2.2 Randomized Algorithms . . . 177
3.2.3 Reduction Between Computational Problems . . . 178
Exercise Set 3.2 . . . 179
3.3 Multiple-precision Integer Arithmetic . . . 180
3.3.1 Representation of Large Integers . . . 181
3.3.2 Basic Arithmetic Operations . . . 181
Addition and subtraction . . . 182
Multiplication . . . 183
Squaring . . . 184
Fast multiplication . . . 184
Division . . . 185
Bit-wise operations . . . 187
3.3.3 GCD . . . 187
3.3.4 Modular Arithmetic . . . 190
Modular exponentiation . . . 190
Montgomery exponentiation . . . 192
Exercise Set 3.3 . . . 193
3.4 Elementary Number-theoretic Computations . . . 195
3.4.1 Primality Testing . . . 195
Deterministic primality proving . . . 197
3.4.2 Generating Random Primes . . . 199

3.4.3 Modular Square Roots . . . 200
Exercise Set 3.4 . . . 201
3.5 Arithmetic in Finite Fields . . . 204
3.5.1 Arithmetic in the Ring $\mathbb{F}_2[X]$ . . . 204
3.5.2 Finite Fields of Characteristic 2 . . . 208
3.5.3 Selecting Suitable Finite Fields . . . 210
3.5.4 Factoring Polynomials over Finite Fields . . . 212
Square-free factorization . . . 212
Distinct-degree factorization . . . 213
Equal-degree factorization . . . 214
Exercise Set 3.5 . . . 215
3.6 Arithmetic on Elliptic Curves . . . 218
3.6.1 Point Arithmetic . . . 218
3.6.2 Counting Points on Elliptic Curves . . . 219
The SEA algorithm . . . 219
The Satoh–FGH algorithm . . . 221
3.6.3 Choosing Good Elliptic Curves . . . 223
3.7 Arithmetic on Hyperelliptic Curves . . . 224
3.7.1 Arithmetic in the Jacobian . . . 225
3.7.2 Counting Points in Jacobians of Hyperelliptic Curves . . . 225
Exercise Set 3.7 . . . 228
3.8 Random Numbers . . . 228
3.8.1 Pseudorandom Bit Generators . . . 228
3.8.2 Cryptographically Strong Pseudorandom Bit Generators . . . 229
3.8.3 Seeding Pseudorandom Bit Generators . . . 230
Exercise Set 3.8 . . . 231

4 The Intractable Mathematical Problems 237
4.1 Introduction . . . 238
4.2 The Problems at a Glance . . . 239
Exercise Set 4.2 . . . 242
4.3 The Integer Factorization Problem . . . 243
4.3.1 Older Algorithms . . . 244
Trial division . . . 244
Pollard's rho method . . . 244
Pollard's $p-1$ method . . . 245
Williams' $p+1$ method . . . 247
4.3.2 The Quadratic Sieve Method . . . 248
The basic algorithm . . . 248
Sieving . . . 249
Incomplete sieving . . . 251
Large prime variation . . . 251
The multiple polynomial quadratic sieve . . . 252
Parallelization . . . 253
TWINKLE: Shamir's factoring device . . . 254
4.3.3 Factorization Using Elliptic Curves . . . 255
4.3.4 The Number Field Sieve Method . . . 258
Selecting the polynomial $f(X)$ . . . 259
Construction of $\mathcal{Q}$ . . . 259

Construction of $\mathcal{G}$ . . . 259
Construction of $\mathcal{U}$ . . . 260
Computing the factorization of $a + b\alpha$ . . . 260
Sieving . . . 261
The running time of the SNFSM . . . 262
Exercise Set 4.3 . . . 262
4.4 The Finite Field Discrete Logarithm Problem . . . 264
4.4.1 Square Root Methods . . . 264
Shanks' baby-step–giant-step method . . . 265
Pollard's rho method . . . 265
The Pohlig–Hellman method . . . 266
4.4.2 The Index Calculus Method . . . 267
4.4.3 Algorithms for Prime Fields . . . 268
The basic ICM . . . 268
The linear sieve method . . . 270
The number field sieve method . . . 272
4.4.4 Algorithms for Fields of Characteristic 2 . . . 273
The basic ICM . . . 274
The adaptation of the linear sieve method . . . 275
Coppersmith's algorithm . . . 276
Exercise Set 4.4 . . . 279
4.5 The Elliptic Curve Discrete Logarithm Problem (ECDLP) . . . 281
4.5.1 The MOV Reduction . . . 282
The correctness of the algorithm . . . 283
Choosing k . . . 284
Computing $e_m(P, R)$ . . . 284
4.5.2 The SmartASS Method . . . 286
4.5.3 The Xedni Calculus Method . . . 289
Exercise Set 4.5 . . . 291
4.6 The Hyperelliptic Curve Discrete Logarithm Problem . . . 292
4.6.1 Choosing the Factor Base . . . 293
4.6.2 Checking the Smoothness of a Divisor . . . 293
4.6.3 The Algorithm . . . 294
4.7 Solving Large Sparse Linear Systems over Finite Rings . . . 294
4.7.1 Structured Gaussian Elimination . . . 296
4.7.2 The Conjugate Gradient Method . . . 297
4.7.3 The Lanczos Method . . . 298
4.7.4 The Wiedemann Method . . . 299
4.8 The Subset Sum Problem . . . 300
4.8.1 The Low-Density Subset Sum Problem . . . 301
4.8.2 The Lattice-Basis Reduction Algorithm . . . 302
Exercise Set 4.8 . . . 304

5 Cryptographic Algorithms 309
5.1 Introduction . . . 310
5.2 Secure Transmission of Messages . . . 310
5.2.1 The RSA Public-key Encryption Algorithm . . . 310
RSA key pair . . . 310
RSA encryption . . . 312

RSA decryption . . . 312
5.2.2 The Rabin Public-key Encryption Algorithm . . . 313
Rabin key pair . . . 313
Rabin encryption . . . 314
Rabin decryption . . . 314
5.2.3 The Goldwasser–Micali Encryption Algorithm . . . 315
Goldwasser–Micali key pair . . . 315
Goldwasser–Micali encryption . . . 315
Goldwasser–Micali decryption . . . 316
5.2.4 The Blum–Goldwasser Encryption Algorithm . . . 317
Blum–Goldwasser key pair . . . 317
Blum–Goldwasser encryption . . . 318
Blum–Goldwasser decryption . . . 318
5.2.5 The ElGamal Public-key Encryption Algorithm . . . 319
ElGamal key pair . . . 319
ElGamal encryption . . . 320
ElGamal decryption . . . 320
5.2.6 The Chor–Rivest Public-key Encryption Algorithm . . . 321
Chor–Rivest key pair . . . 321
Chor–Rivest encryption . . . 322
Chor–Rivest decryption . . . 322
5.2.7 The XTR Public-key Encryption Algorithm . . . 323
XTR key pair . . . 327
XTR encryption . . . 327
XTR decryption . . . 328
5.2.8 The NTRU Public-key Encryption Algorithm . . . 328
NTRU key pair . . . 328
NTRU encryption . . . 330
NTRU decryption . . . 331
Exercise Set 5.2 . . . 332
5.3 Key Exchange . . . 334
5.3.1 Basic Key-Exchange Protocols . . . 334
The Diffie–Hellman key-exchange protocol . . . 334
Small-subgroup attacks . . . 335
Cofactor exponentiation . . . 336
5.3.2 Authenticated Key-Exchange Protocols . . . 336
Unknown key-share attacks . . . 336
The Menezes–Qu–Vanstone key-exchange protocol . . . 338
Exercise Set 5.3 . . . 339
5.4 Digital Signatures . . . 340
5.4.1 The RSA Digital Signature Algorithm . . . 341
5.4.2 The Rabin Digital Signature Algorithm . . . 342
5.4.3 The ElGamal Digital Signature Algorithm . . . 343
5.4.4 The Schnorr Digital Signature Algorithm . . . 344
5.4.5 The Nyberg–Rueppel Digital Signature Algorithm . . . 345
5.4.6 The Digital Signature Algorithm (DSA) . . . 346
5.4.7 The Elliptic Curve Digital Signature Algorithm (ECDSA) . . . 348
5.4.8 The XTR Signature Algorithm . . . 349
5.4.9 The NTRUSign Algorithm . . . 352

5.4.10 Blind Signature Schemes . . . 355
Chaum's RSA blind signature protocol . . . 355
The Schnorr blind signature protocol . . . 356
The Okamoto–Schnorr blind signature protocol . . . 357
5.4.11 Undeniable Signature Schemes . . . 357
The Chaum–Van Antwerpen undeniable signature scheme . . . 358
RSA-based undeniable signature scheme . . . 360
5.4.12 Signcryption . . . 362
Exercise Set 5.4 . . . 364
5.5 Entity Authentication . . . 366
5.5.1 Passwords . . . 366
5.5.2 Challenge–Response Algorithms . . . 368
A challenge–response scheme based on encryption–decryption . . . 368
A challenge–response scheme based on digital signatures . . . 369
Mutual authentication . . . 370
5.5.3 Zero-Knowledge Protocols . . . 370
The Feige–Fiat–Shamir (FFS) protocol . . . 372
The Guillou–Quisquater (GQ) protocol . . . 373
The Schnorr protocol . . . 374
Exercise Set 5.5 . . . 375

6 Standards **381**
6.1 Introduction . . . 382
6.2 IEEE Standards . . . 382
6.2.1 The Data Types . . . 383
Bit strings . . . 383
Octet strings . . . 383
Integers . . . 384
Prime finite fields . . . 384
Finite fields of characteristic 2 . . . 384
Extension fields of odd characteristics . . . 384
Elliptic curves . . . 385
Elliptic curve points . . . 385
Convolution polynomial rings . . . 386
6.2.2 Conversion Among Data Types . . . 386
Converting bit strings to octet strings (BS2OS) . . . 386
Converting octet strings to bit strings (OS2BS) . . . 387
Converting integers to bit strings (I2BS) . . . 388
Converting bit strings to integers (BS2I) . . . 388
Converting integers to octet strings (I2OS) . . . 388
Converting octet strings to integers (OS2I) . . . 389
Converting field elements to octet strings (FE2OS) . . . 389
Converting octet strings to field elements (OS2FE) . . . 389
Converting field elements to integers (FE2I) . . . 389
Converting elliptic curve points to octet strings (EC2OS) . . . 389
Converting octet strings to elliptic curve points (OS2EC) . . . 390
Converting ring elements to octet strings (RE2OS) . . . 390
Converting octet strings to ring elements (OS2RE) . . . 391
Converting ring elements to bit strings (RE2BS) . . . 391

Converting bit strings to ring elements (BS2RE) . . . 391
Converting binary elements to octet strings (BE2OS) . . . 392
Converting octet strings to binary elements (OS2BE) . . . 392
6.3 RSA Standards . . . 393
6.3.1 PKCS #1 . . . 393
RSA keys . . . 394
RSA key operations . . . 394
RSAES–OAEP encryption scheme . . . 395
RSASSA–PSS signature scheme with appendix . . . 398
A mask-generation function . . . 399
The RSA encryption scheme of PKCS #1, Version 1.5 . . . 400
The RSA signature scheme of PKCS #1, Version 1.5 . . . 401
6.3.2 PKCS #3 . . . 402

7 Cryptanalysis in Practice 405
7.1 Introduction . . . 406
7.2 Side-Channel Attacks . . . 407
7.2.1 Timing Attack . . . 407
Details of the attack . . . 407
Countermeasures . . . 410
7.2.2 Power Analysis . . . 411
Simple power analysis (SPA) . . . 411
Differential power analysis (DPA) . . . 413
Countermeasures . . . 415
7.2.3 Fault Analysis . . . 416
Fault attack on RSA based on CRT . . . 417
Fault attack on RSA without CRT . . . 417
Fault attack on the Rabin digital signature algorithm . . . 418
Fault attack on DSA . . . 418
Fault attack on the ElGamal signature scheme . . . 419
Fault attack on the Feige–Fiat–Shamir identification protocol . . . 420
Countermeasures . . . 422
Exercise Set 7.2 . . . 423
7.3 Backdoor Attacks . . . 424
7.3.1 Attacks on RSA . . . 425
Hiding prime factor . . . 425
Hiding small private exponent . . . 428
Hiding small public exponent . . . 429
7.3.2 An Attack on ElGamal Signatures . . . 430
7.3.3 An Attack on ElGamal Encryption . . . 431
7.3.4 Countermeasures . . . 432
Exercise Set 7.3 . . . 432

8 Quantum Computation and Cryptography 437
8.1 Introduction . . . 438
8.2 Quantum Computation . . . 438
8.2.1 System . . . 439
8.2.2 Entanglement . . . 440
8.2.3 Evolution . . . 442
8.2.4 Measurement . . . 443

8.2.5 The Deutsch Algorithm . . . 445
Exercise Set 8.2 . . . 446
8.3 Quantum Cryptography . . . 448
Exercise Set 8.3 . . . 451
8.4 Quantum Cryptanalysis . . . 452
8.4.1 Shor's Algorithm for Computing Period . . . 453
8.4.2 Breaking RSA . . . 455
8.4.3 Factoring Integers . . . 455
8.4.4 Computing Discrete Logarithms . . . 456
Exercise Set 8.4 . . . 458

Appendices

A Symmetric Techniques **465**
A.1 Introduction . . . 466
A.2 Block Ciphers . . . 466
A.2.1 A Case Study: DES . . . 467
DES key schedule . . . 468
DES encryption . . . 468
DES decryption . . . 471
DES test vectors . . . 471
Cryptanalysis of DES . . . 471
A.2.2 The Advanced Standard: AES . . . 472
Data representation . . . 472
AES key schedule . . . 473
AES encryption . . . 474
AES decryption . . . 476
AES test vectors . . . 478
Cryptanalysis of AES . . . 478
A.2.3 Multiple Encryption . . . 478
A.2.4 Modes of Operation . . . 480
The ECB mode . . . 480
The CBC mode . . . 481
The CFB mode . . . 481
The OFB mode . . . 482
Exercise Set A.2 . . . 483
A.3 Stream Ciphers . . . 486
A.3.1 Linear Feedback Shift Registers . . . 487
A.3.2 Stream Ciphers Based on LFSRs . . . 489
Exercise Set A.3 . . . 490
A.4 Hash Functions . . . 491
A.4.1 Merkle's Meta Method . . . 492
A.4.2 The Secure Hash Algorithm . . . 494
Exercise Set A.4 . . . 495

B Key Exchange in Sensor Networks **497**
B.1 Introduction . . . 498
B.2 Security Issues in a Sensor Network . . . 498
B.3 The Basic Bootstrapping Framework . . . 500
B.4 The Basic Random Key Predistribution Scheme . . . 502

B.4.1 The q-composite Scheme . . . 504
B.4.2 Multi-path Key Reinforcement . . . 505
B.5 Random Pairwise Scheme . . . 506
B.5.1 Multi-hop Range Extension . . . 507
B.6 Polynomial-pool-based Key Predistribution . . . 508
B.6.1 Pairwise Key Predistribution . . . 509
B.6.2 Grid-based Key Predistribution . . . 510
B.7 Matrix-based Key Predistribution . . . 511
B.8 Location-aware Key Predistribution . . . 513
B.8.1 Closest Pairwise Keys Scheme . . . 513
B.8.2 Location-aware Polynomial-pool-based Scheme . . . 515

C Complexity Theory and Cryptography **517**
C.1 Introduction . . . 518
C.2 Provably Difficult Computational Problems Are not Suitable . . . 519
Exercise Set C.2 . . . 519
C.3 One-way Functions and the Complexity Class UP . . . 520
Exercise Set C.3 . . . 522

D Hints to Selected Exercises **523**

References **531**

Index **547**

Preface

I can't understand why a person will take a year to write a novel
when he can easily buy one for a few dollars.
— Fred Allen

The first moral question that we faced (like most authors) is: "Why another book?" Available textbooks on public-key cryptography (or cryptography in general) are many [37, 74, 113, 114, 145, 152, 153, 194, 209, 262, 283, 288, 291, 296]. In the presence of all these books, writing another may sound like a waste of energy and effort.

Fortunately, we have a big answer. Most cryptography textbooks today, even many of the celebrated ones, essentially take a narrative approach. While such an approach may be suitable for beginners at an undergraduate level, it misses the finer details in this rapidly growing area of applied mathematics. The fact that public-key cryptography *is* mathematical is hard to deny and a mathematical subject would be better treated in the mathematical way.

This is precisely the point that this book addresses, that is, it proceeds in a canonically mathematical way while revealing cryptographic concepts. This mathematics is often not so simple (and that is why other textbooks didn't bother to mention it), but we plan to stick to mathematical sophistication as far as possible. A typical feature of this book is that it does not rely on anything other than the readers' mathematical intuitions; it develops all the mathematical abstractions starting from scratch. Although computer science and mathematics students nowadays do undergo some courses on discrete structures somewhere in their curricula, we do not assume this; instead we develop the algebra starting at the level of set operations. Simpler structures like groups, rings and fields are followed by more complex concepts like finite fields, algebraic curves, number fields and p-adic numbers. The resulting (long) compilation of abstract mathematical tools tends to relieve cryptography students and researchers from consulting many mathematics books for understanding the background concepts. We are happy to offer this self-sufficient treatment complete with proofs and other details. The only place where we had to be somewhat sketchy is the discussion on elliptic and hyperelliptic curves. The mathematics here seems to be too vast to fit in a few pages and we opted for a deliberate simplification of these topics.

A big problem with discrete mathematics is that many of its proofs are existential. However, in order to make things work in a practical environment one must undergo algorithmic studies of algebra and number theory. This is what our book does next. While many algorithmic issues in this area are settled favourably, there remain some problems whose best known algorithmic complexities are still poor. Some of these so-called computationally difficult problems are used to build *secure* public-key cryptosystems. The security of these systems are assumed (rather than proven) and so we extensively deal with the algorithms known till date to solve these difficult problems. This is precisely the point that utilizes the mathematics developed in earlier chapters, to a great extent.

In Chapter 5, we eventually hit upon the culmination of all these mathematical and algorithmic studies in the design of public-key systems for achieving various cryptographic goals. Under the theoretical base developed in earlier chapters, Chapter 5 turns out to be an easy chapter. This is our way of looking into the problem, namely, a formal bottom–up approach. We claim to be different from most textbooks in this regard. Our discussion of mathematics is not for its own sake, but to develop the foundation of cryptographic primitives.

We then turn to some purely implementation and practical issues of public-key cryptography. Standards proposed by organizations such as IEEE and RSA Security Inc. promote interoperability of using crypto primitives in Internet applications. We then look at some small applications of the crypto basics. Some indirect ways of cryptanalysis are described next. These techniques (side-channel and backdoor attacks) give the book a strong practical flavour in tandem with its otherwise formal appearance.

As an eleventh-hour decision, we added a final chapter to our book, a chapter on quantum computation and its implications on public-key cryptography. Although somewhat theoretical at this point, quantum

computation exhibits important ramifications in public-key cryptography. The mathematics behind quantum mechanics and computation are never discussed earlier just to highlight the distinctive nature of this chapter, which may perhaps be titled *cryptography in future*.

This schematic description of this book perhaps makes it clear that this book is better suited as a graduate-level textbook. A one- or two-semester graduate or advanced undergraduate course can run based on the contents of this book. Self-studying this book is also possible at an advanced graduate or research level, but is expected to be difficult at an undergraduate level. We highlight the importance of classroom teaching, if an undergraduate course is to be based on this textbook.

We rated different items in the book by their levels of difficulty and/or mathematical sophistication. Unstarred items can be covered even in undergraduate courses. Items marked by single stars can be taken seriously for a second course or a second reading. Doubly starred items, on the other hand, are research-level materials and can be pursued only in really advanced courses or for undergoing research. Inclusion of a good amount of these advanced topics marks another distinction of this book compared to other available textbooks.

The book comes with plenty of exercises. We have two-fold motivations behind these exercises. In the first place, they help the readers deepen their understanding of the matter discussed in the text. In the second place, some of these exercises build additional theory that we omit in the text proper. We occasionally make use of these additional topics in proving and/or explaining results in the text. We do not classify the exercises into easy and difficult ones, but specify hints, some of which are pretty explicit, for intellectually challenging parts. We separate out the hints in an appendix near the end of this book and leave the marker **[H]** in appropriate locations of the statements of the exercises. This practice prevents a reader from accidentally seeing a hint. Only when the reader gets stuck, (s)he can look at the hints at the end. We believe that the exercises, together with our discussion on algorithms and implementation issues, will offer serious students many ways to carry out substantial implementation work to further their research and development in cryptography.

Every chapter ends with annotated references for further studies. We do not claim to be encyclopaedic in this respect. Instead we mention only those references that, we feel, are directly related to the topics dealt with in the respective chapters.

As a trade-off between bulk and coverage, we had to leave many issues untouched. For example, we were limited by constraints of space to present symmetric-key cryptography in detail. However, in view of its importance today, we include brief discussions in an appendix on block ciphers, stream ciphers and hash functions. We also do not discuss anything about formal security of public-key protocols. The issues related to *provable security* are at the minimum theoretically important in the study of cryptography, but are entirely left out here. Only a brief discussion on the implication of complexity theory on the security of public-key protocols is included in another appendix. The *Handbook of Applied Cryptography* [194] by Menezes et al. can supplement this book for learning symmetric techniques, whereas the book by Delfs and Knebl [74] or those by Goldreich [113, 114] can be consulted for formal security issues.

We are indebted to everybody whose criticism, encouragement and support made this project materializable. Special thanks go to Bimal Roy, Chandan Mazumdar, C. Pandurangan, Debdeep Mukhopadhyay, Dipanwita Roychowdhury, Gagan Garg, Hartmut Wiebe, H. V. Kumar Swamy, Indranil Sengupta, Kapil Paranjape, Manindra Agarwal, Palash Sarkar, Rajesh Pillai, Rana Barua, R. Balasubramanian, Sanjay Barman, Shailesh, Satrajit Ghosh, Souvik Bhattacherjee, Srihari Vavilapalli, Subhamoy Maitra, Surjyakanta Mohapatro, and Uwe Storch. This book has been tested in postgraduate courses in the Indian Institute of Science, Bangalore, and in the Indian Institute of Technology Kharagpur. We sincerely thank all our students for pointing out many errors and suggesting several improvements. We express our deep gratitude to our family members for their constant understanding and moral support. We are also indebted to our institutes for providing the wonderful intellectual climate for completing this work.

A. D.
C. E. V. M.

Notations

Any time you are stuck on a problem, introduce more notation.
— Chris Skinner [Plenary Lecture, Aug 1997, Topics in Number Theory, Penn State]

General

$\lvert a\rvert$	absolute value of real number a
$\min S$	minimum of elements of set S
$\max S$	maximum of elements of set S
$\exp(a)$	e^a, where $e = \sum_{n=0}^{\infty} 1/n! = 2.7182818284\ldots$
$\log x$	logarithm of x with respect to some unspecified base (like 10)
$\ln x$	$\log_e x$, where $e = \sum_{n=0}^{\infty} 1/n!$
$\lg x$	$\log_2 x$
$\log^k x$	$(\log x)^k$ (similarly, $\ln^k x = (\ln x)^k$ and $\lg^k x = (\lg x)^k$)
$:=$	is defined as (or "is assigned the value" in code snippets)
i	$\sqrt{-1}$
$\bar{z}$	complex conjugate $(x - \mathrm{i}y)$ of the complex number $z = x + \mathrm{i}y$
δ_{ij}	Kronecker delta
$(a_s a_{s-1} \ldots a_0)_b$	b-ary representation of a non-negative integer
$\binom{n}{r}$	binomial coefficient, equals $n(n-1)\cdots(n-r+1)/r!$
$\lfloor x \rfloor$	floor of real number x
$\lceil x \rceil$	ceiling of real number x
$[a, b]$	closed interval, that is, the set of real numbers x in the range $a \leqslant x \leqslant b$
(a, b)	open interval, that is, the set of real numbers x in the range $a < x < b$
$L(t, \alpha, c)$	expression of the form $\exp\big((c + \mathrm{o}(1))(\ln t)^{\alpha}(\ln\ln t)^{1-\alpha}\big)$
$L_t[c]$	abbreviation for $L(t, 1/2, c)$ (denoted also as $L[c]$ if t is understood)

Bit-wise operations (on bit strings a, b)

NAND	negation of AND
NOR	negation of OR
XOR	exclusive OR
$a \oplus b$	bit-wise exclusive OR (XOR) of a and b
a AND b	bit-wise AND of a and b
a OR b	bit-wise inclusive OR of a and b
$\mathrm{LS}_k(a)$	left shift of a by k bits
$\mathrm{RS}_k(a)$	right shift of a by k bits
$\mathrm{LR}_k(a)$	left rotate (cyclic left shift) of a by k bits
$\mathrm{RR}_k(a)$	right rotate (cyclic right shift) of a by k bits
$\overline{a}$	bit-wise complement of a
$a \mid\mid b$	concatenation of a and b

Sets

$\emptyset$	empty set
$\#A$	cardinality of set A
$a \in A$	a is an element of set A
$A \subseteq B$	set A is contained in set B
$A \not\subseteq B$	set A is not contained in set B
$A \subsetneq B$	set A is properly contained in set B
$A \cup B$	union of sets A and B
$A \uplus B$	disjoint union of sets A and B

$A \cap B$	intersection of sets A and B
$A \setminus B$	difference of sets A and B
$\bar{A}$	complement of set A (in a bigger set)
$A \times B$	(Cartesian) product of sets A and B
$\mathbb{N}$	set of all natural numbers, that is, $\{1, 2, 3, \ldots\}$
$\mathbb{N}_0$	set of all non-negative integers, that is, $\{0, 1, 2, \ldots\}$
$\mathbb{Z}$	set of all integers, that is, $\{\ldots, -2, -1, 0, 1, 2, \ldots\}$
$\mathbb{P}$	set of all (positive) prime numbers, that is, $\{2, 3, 5, 7, \ldots\}$
$\mathbb{Q}$	set of all rational numbers, that is, $\{a/b \mid a \in \mathbb{Z}, b \in \mathbb{N}\}$
$\mathbb{Q}^*$	set of all non-zero rational numbers
$\mathbb{R}$	set of all real numbers
$\mathbb{R}^*$	set of all non-zero real numbers
$\mathbb{R}_+$	set of all non-negative real numbers
$\mathbb{C}$	set of all complex numbers
$\mathbb{C}^*$	set of all non-zero complex numbers
$\mathbb{Z}_n$	$\mathbb{Z}/n\mathbb{Z}$, can be represented by the set $\{0, 1, \ldots, n-1\}$
$\mathbb{Z}_n^*$	group of units in $\mathbb{Z}_n$, can be represented as $\{a \mid 0 \leqslant a < n, \gcd(a, n) = 1\}$
$\mathbb{F}_q$	finite field of cardinality q
$\mathbb{F}_q^*$	multiplicative group of $\mathbb{F}_q$, that is, $\mathbb{F}_q \setminus \{0\}$
$\mathfrak{O}_K$	ring of integers of number field K
$\mathfrak{U}_K$	group of units of $\mathfrak{O}_K$
$\hat{\mathbb{Z}}_p$	ring of p-adic integers
$\mathbb{Q}_p$	field of p-adic numbers
U_p	group of units of $\hat{\mathbb{Z}}_p$

Functions and relations

$f : A \to B$	f is a function from set A to set B
$f : A \hookrightarrow B$	f is an injective function from set A to set B
$f : A \twoheadrightarrow B$	f is a surjective function from set A to set B
$a \mapsto b$	a is mapped to b (by a function)
$f \circ g$	composition of functions f and g (applied from right to left)
f^{-1}	inverse of bijective function f
$\operatorname{Ker} f$	kernel of function (homomorphism) f
$\operatorname{Im} f$	image of function f
$\sim$	equivalent to
$[a]$	equivalence class of a

Groups

aH	coset in a multiplicative group
$a + H$	coset in an additive group
HK	internal direct product of (sub)groups H and K
$H \times K$	external direct product of (sub)groups H and K
$[G : H]$	index of subgroup H in group G
G/H	quotient group
$G_1 \cong G_2$	groups G_1 and G_2 are isomorphic
$\operatorname{ord} G$	order (that is, cardinality) of group G
$\operatorname{ord}_G a$	order of element a in group G
$\operatorname{Exp} G$	exponent of group G
$Z(G)$	centre of group G
$C(a)$	centralizer of group element a

$GL_n(K)$	general linear group over field K (of $n \times n$ matrices)
$SL_n(K)$	special linear group over field K (of $n \times n$ matrices)
G_{tors}	torsion subgroup of G

Rings

$\mathrm{char}\, A$	characteristic of ring A
$A \times B$	direct product of rings A and B
A^*	multiplicative group of units of ring A
$\langle S \rangle$	for ring A, ideal generated by $S \subseteq A$, also written as $\sum_{a \in S} aA$
$\langle a \rangle$	for ring A, principal ideal generated by $a \in A$, also written as aA and Aa
$a \equiv b \pmod{\mathfrak{a}}$	a is congruent to b modulo ideal $\mathfrak{a}$, that is, $a - b \in \mathfrak{a}$
$A \cong B$	rings A and B are isomorphic
$A/\mathfrak{a}$	quotient ring (modulo ideal $\mathfrak{a}$)
$a \mid b$	a divides b (in some ring)
$\mathrm{v}_p(a)$	multiplicity of prime p in element a
$p^k \mid\mid a$	$k = \mathrm{v}_p(a)$
$\mathfrak{n}_A$	nilradical of ring A
A_{red}	reduction of ring A, equals $A/\mathfrak{n}_A$
$\gcd(a, b)$	greatest common divisor of elements a and b
$\mathrm{lcm}(a, b)$	least common multiple of elements a and b
$\mathfrak{a} + \mathfrak{b}$	sum of ideals $\mathfrak{a}$ and $\mathfrak{b}$
$\mathfrak{a} \cap \mathfrak{b}$	intersection of ideals $\mathfrak{a}$ and $\mathfrak{b}$
$\mathfrak{a}\mathfrak{b}$	product of ideals $\mathfrak{a}$ and $\mathfrak{b}$
$\sqrt{\mathfrak{a}}$	root (or radical) of ideal $\mathfrak{a}$
$\mathrm{Q}(A)$	total quotient ring of ring A (quotient field of A, if A is an integral domain)
$S^{-1}A$	localization of ring A at multiplicative set S
$A_{\mathfrak{p}}$	localization of ring A at prime ideal $\mathfrak{p}$
$\mathfrak{O}_K$	ring of integers of number field K
$\mathrm{N}(\mathfrak{a})$	norm of ideal $\mathfrak{a}$ (in a Dedekind domain)
CRT	Chinese remainder theorem
ED	Euclidean domain
DD	Dedekind domain
DVD (or DVR)	discrete valuation domain (or ring)
PID	principal ideal domain
UFD	unique factorization domain

Fields

$\mathrm{char}\, K$	characteristic of field K
K^*	multiplicative group of units of field K, that is, $K \setminus \{0\}$
$\bar{K}$	algebraic closure of field K
$[K : F]$	degree of the field extension $F \subseteq K$
$K[a]$	$\{f(a) \mid f(X) \in K[X]\}$
$K(a)$	$\{f(a)/g(a) \mid f(X), g(X) \in K[X], g(a) \neq 0\}$
$\mathrm{Aut}\, K$	group of automorphisms of field K
$\mathrm{Aut}_F\, K$	for field extension $F \subseteq K$, group of F-automorphisms of K (also $\mathrm{Gal}(K\|F)$)
$\mathrm{Fix}_F\, H$	for field extension $F \subseteq K$, fixed field of subgroup H of $\mathrm{Aut}_F\, K$
$\mathbb{F}_q$	finite field of cardinality q
$\mathbb{F}_q^*$	multiplicative group of units of $\mathbb{F}_q$, that is, $\mathbb{F}_q \setminus \{0\}$
Tr	trace function
$\mathrm{Tr}_{K\|F}(a)$	for field extension $F \subseteq K$, trace of $a \in K$ over F
N	norm function

$\mathrm{N}_{K\|F}(a)$	for field extension $F \subseteq K$, norm of $a \in K$ over F
$\mathfrak{F}_q$	Frobenius automorphism $\mathbb{F}_{q^m} \to \mathbb{F}_{q^m}$, $a \mapsto a^q$
$\mathfrak{O}_K$	ring of integers of number field K
$\mathfrak{U}_K$	group of units of $\mathfrak{O}_K$
Δ_K	discriminant of number field K
$\hat{\mathbb{Z}}_p$	ring of p-adic integers
$\mathbb{Q}_p$	field of p-adic numbers
U_p	group of units of $\hat{\mathbb{Z}}_p$
$\|\ \|_p$	p-adic norm on $\mathbb{Q}_p$

Integers

$a \operatorname{quot} b$	quotient of Euclidean division of a by $b \neq 0$
$a \operatorname{rem} b$	remainder of Euclidean division of a by $b \neq 0$
$a\|b$	a divides b in $\mathbb{Z}$, that is, $b = ca$ for some $c \in \mathbb{Z}$
$\mathrm{v}_p(a)$	multiplicity of prime p in non-zero integer a
$\gcd(a, b)$	greatest common divisor of integers a and b (not both zero)
$\operatorname{lcm}(a, b)$	least common multiple of integers a and b
$a \equiv b \pmod{n}$	a is congruent to b modulo n
$a^{-1} \pmod{n}$	multiplicative inverse of a modulo n (given that $\gcd(a, n) = 1$)
$\phi(n)$	Euler's totient function
$\left(\frac{a}{p}\right)$	Legendre (or Jacobi) symbol
$[a]_n$	coset $a + n\mathbb{Z} = \{a + kn \mid k \in \mathbb{Z}\}$
$\operatorname{ord}_n a$	multiplicative order of a modulo n (given that $\gcd(a, n) = 1$)
$\mu(n)$	Möbius function
$\pi(x)$	number of primes between 1 and positive real number x
$\mathrm{Li}(x)$	Gauss' Li function
$\psi(x, y)$	fraction of positive integers $\leqslant x$, that are y-smooth
$\zeta(s)$	Riemann zeta function
RH	Riemann hypothesis
ERH	extended Riemann hypothesis
M_n	$2^n - 1$ (Mersenne number)
$\mathfrak{R}$	2^{32}, standard radix for representation of multiple-precision integers

Polynomials

$A[X_1, \ldots, X_n]$	polynomial ring in indeterminates $X_1, \ldots, X_n$ over ring A
$A(X_1, \ldots, X_n)$	ring of rational functions in indeterminates $X_1, \ldots, X_n$ over ring A
$\deg f$	degree of polynomial f
$\mathrm{lc}\, f$	leading coefficient of polynomial f
$\mathrm{minpoly}_{\alpha,K}(X)$	minimal polynomial of α over field K, belongs to $K[X]$
$\mathrm{cont}\, f$	content of polynomial f
$\mathrm{pp}\, f$	primitive part of polynomial f
$f'(X)$	formal derivative of polynomial $f(X)$
$\Delta(f)$	discriminant of polynomial f
$\mathcal{I}_{q,m}(X)$	the polynomial $X^{q^m} - X \in \mathbb{F}_q[X]$
μ_m	group of m-th roots of unity
Φ_m	m-th cyclotomic polynomial

Vector spaces, modules and matrices

$\dim_K V$	dimension of vector space V over field K
$\mathrm{Span}\, S$	span of subset S of a vector space

$\mathrm{Hom}_K(V, W)$	set of all K-linear transformations $V \to W$
$\mathrm{End}_K(V)$	set of all K-linear transformations $V \to V$
M/N	quotient vector space or module
$M \cong N$	vector spaces or modules M and N are isomorphic
$\prod_{i\in I} M_i$	direct product of modules M_i, $i \in I$
$\bigoplus_{i\in I} M_i$	direct sum of modules M_i, $i \in I$
A^{t}	transpose of matrix (or vector) A
A^{-1}	inverse of matrix A
$\mathrm{Rank}\, T$	rank of matrix or linear transformation T
$\mathrm{Rank}_A M$	rank of A-module M
$\mathrm{Null}\, T$	nullity of matrix or linear transformation T
$(M : N)$	for A-module M and submodule N, the ideal $\{a \in A \mid aM \subseteq N\}$ of A
$\mathrm{Ann}_A(M)$	annihilator of A-module M, same as $(M : 0)$
$\mathrm{Tors}\, M$	torsion submodule of M
$A[S]$	A-algebra generated by set S
$\langle \mathbf{v}, \mathbf{w} \rangle$	inner product of two real vectors **v** and **w**

Algebraic curves

$\mathbb{A}^n(K)$	n-dimensional affine space over field K
$\mathbb{P}^n(K)$	n-dimensional projective space over field K
$(x_1, \ldots, x_n)$	homogeneous coordinates of a point in $\mathbb{A}^n(K)$
$[x_0, x_1, \ldots, x_n]$	projective coordinates of a point in $\mathbb{P}^n(K)$
$f^{(h)}$	homogenization of polynomial f
$C(K)$	set of K-rational points over curve C defined over field K
$K[C]$	ring of polynomial functions on curve C defined over K
$K(C)$	field of rational functions on curve C defined over K
$[P]$	point P on a curve in formal sums
$\mathrm{ord}_P(r)$	order of rational function r at point P
$\mathrm{Div}_K(C)$	group of divisors on curve C defined over field K
$\mathrm{Div}^0_K(C)$	group of divisors of degree 0 on curve C defined over field K
$\mathrm{Div}_K(r)$	divisor of a rational function r
$\mathrm{Prin}_K(C)$	group of principal divisors on curve C defined over field K
$\mathbb{J}_K(C)$	Jacobian of curve C defined over field K
$\mathrm{Pic}_K(C)$	Picard group of curve C (equals $\mathrm{Div}_K(C)/\mathrm{Prin}_K(C)$)
$\mathrm{Pic}^0_K(C)$	$\mathrm{Div}^0_K(C)/\mathrm{Prin}_K(C)$, same as Jacobian $\mathbb{J}_K(C)$
$\mathcal{O}$	point at infinity on an elliptic or a hyperelliptic curve
$\Delta(E)$	discriminant of elliptic curve E
$j(E)$	j-invariant of elliptic curve E
$E(K)$	group of points on elliptic curve E defined over field K
$P + Q$	sum of two points $P, Q \in E(K)$
mP	m-th multiple (that is, m-fold sum) of point $P \in E(K)$
ψ_m, $\bar{f}_m$, f_m	m-th division polynomials
t	trace of Frobenius of elliptic curve ($\#E(\mathbb{F}_q) = q + 1 - t$)
$E_K[m]$	group of m-torsion points in $E(K)$
$E[m]$	abbreviation for $E_{\bar{K}}[m]$
e_m	Weil pairing (a map $E[m] \times E[m] \to \mu_m$)
$\mathrm{Div}(a, b)$	representation of reduced divisor on hyperelliptic curve by polynomials a, b

Probability and statistics

$\Pr(E)$	probability of event E
$\Pr(E_1 \mid E_2)$	conditional probability of event E_1 given event E_2

$\mathrm{E}(X)$	expectation of random variable X
$\mathrm{Var}(X)$	variance of random variable X
σ_X	standard deviation of random variable X (equals $\sqrt{\mathrm{Var}(X)}$)
$\mathrm{Cov}(X, Y)$	covariance of random variables X, Y
$\rho_{X,Y}$	correlation coefficient of random variables X, Y

Computational complexity

$f = \mathrm{O}(g)$	big-Oh notation: f is of the order of g
$f = \Omega(g)$	big-Omega notation: g is of the order of f
$f = \Theta(g)$	big-Theta notation: f and g have the same order
$f = \mathrm{o}(g)$	small-oh notation: f is of strictly smaller order than g
$f = \omega(g)$	small-omega notation: f is of strictly larger order than g
$f = \mathrm{O}\tilde{}(g)$	soft-Oh notation: $f = \mathrm{O}(g \log^k g)$ for real constant $k \geqslant 0$
$P_1 \preceq P_2$	problem P_1 is polynomial-time reducible to problem P_2
$P_1 \cong P_2$	problems P_1 and P_2 are polynomial-time equivalent

Intractable problems

CVP	closest vector problem
DHP	(finite field) Diffie–Hellman problem
DLP	(finite field) discrete logarithm problem
ECDHP	elliptic curve Diffie–Hellman problem
ECDLP	elliptic curve discrete logarithm problem
HECDHP	hyperelliptic curve Diffie–Hellman problem
HECDLP	hyperelliptic curve discrete logarithm problem
GIFP	general integer factorization problem
IFP	integer factorization problem
QRP	quadratic residuosity problem
RSAIFP	RSA integer factorization problem
RSAKIP	RSA key inversion problem
RSAP	RSA problem
SQRTP	modular square root problem
SSP	subset sum problem
SVP	shortest vector problem

Algorithms

ADH	Adleman, DeMarrais and Huang's algorithm
AES	advanced encryption standard
AKS	Agarwal, Kayal and Saxena's deterministic primality test
BSGS	Shanks' baby-step–giant-step method
CBC	cipher-block chaining mode
CFB	cipher feedback mode
CSM	cubic sieve method
CSPRBG	cryptographically strong pseudorandom bit generator
CvA	Chaum and Van Antwerpen's undeniable signature scheme
DDF	distinct-degree factorization
DES	data encryption standard
DH	Diffie–Hellman key exchange
DPA	differential power analysis
DSA	digital signature algorithm
DSS	digital signature standard
ECB	electronic codebook mode

ECDSA	elliptic curve digital signature algorithm
ECM	elliptic curve method
E-D-E	encryption–decryption–encryption scheme of triple encryption
EDF	equal-degree factorization
EG	Eschenauer and Gligor's scheme
FEAL	fast data encipherment algorithm
FFS	Feige, Fiat and Shamir's zero-knowledge protocol
GKR	Gennaro, Krawczyk and Rabin's RSA-based undeniable signature scheme
GNFSM	general number field sieve method
GQ	Guillou and Quisquater's zero-knowledge protocol
HFE	cryptosystem based on hidden field equations
ICM	index calculus method
IDEA	international data encryption algorithm
KLCHKP	braid group cryptosystem
L^3	Lenstra–Lenstra–Lovasz algorithm
LFSR	linear feedback shift register
LSM	linear sieve method
LUC	cryptosystem based on Lucas sequences
MOV	Menezes, Okamoto and Vanstone's reduction
MPQSM	multiple polynomial quadratic sieve method
MQV	Menezes–Qu–Vanstone key exchange
NFSM	number field sieve method
NR	Nyberg–Rueppel signature algorithm
NTRU	Hoffstein, Pipher and Silverman's encryption algorithm
NTRUSign	NTRU signature algorithm
OAEP	optimal asymmetric encryption procedure
OFB	output feedback mode
PAP	pretty awful privacy
PGP	pretty good privacy
PH	Pohlig–Hellman method
PRBG	pseudorandom bit generator
PSS	probabilistic signature scheme
QSM	quadratic sieve method
RSA	Rivest, Shamir and Adleman's algorithm
SAFER	secure and fast encryption routine
Satoh–FGH	Point counting algorithm on elliptic curves over fields of characteristic 2
SDSA	shortened digital signature algorithm
SEA	Schoof, Elkies and Atkins' algorithm for point counting on elliptic curves
SETUP	secretly embedded trapdoor with universal protection
SFF	square-free factorization
SHA	secure hash algorithm
SmartASS	algorithm for computing discrete logs in anomalous elliptic curves
SNFSM	special number field sieve method
SPA	simple power analysis
TWINKLE	the Weizmann Institute key location engine
TWIRL	the Weizmann Institute relation locator
XCM	xedni calculus method
XSL	extended sparse linearization attack
XTR	efficient and compact subgroup trace representation

ZK	zero-knowledge

Quantum computation

$\|\psi\rangle$	ket notation for vector ψ
$\langle\psi\|\varphi\rangle$	inner product of vectors $\|\psi\rangle$ and $\|\varphi\rangle$
$\|\|\psi\|\|$	norm of vector $\|\psi\rangle$ (equals $\sqrt{\langle\psi\|\psi\rangle}$)
$\mathcal{H}_n$	n-dimensional Hilbert space (over $\mathbb{C}$)
$\|0\rangle, \|1\rangle, \ldots, \|n-1\rangle$	orthonormal basis of $\mathbb{C}^n$
cbit	classical bit
qubit	quantum bit
$\otimes$	tensor product of Hilbert spaces
F	Fourier transform
H	Hadamard transform
I	Identity transform
X	Exchange transform
Z	Z transform

Computational primitives

ulong	32-bit unsigned integer data type (unsigned long)
ullong	64-bit unsigned integer data type (unsigned long long)
$a := b$	assignment operator (returns the value assigned)
$+, -, \times, /, \%$	arithmetic operators
$++, --$	increment and decrement operators
$a \diamond= b$	$a := a \diamond b$ for $\diamond \in \{+, -, \times, /, \%\}$
$=, \neq, >, <, \geqslant, \leqslant$	comparison operators
1	*True* as a condition
if	conditional statement: if (condition) $\cdots$
if-else	conditional statement: if (condition) $\cdots$, else $\cdots$
while	while loop: while (condition) $\cdots$
do	do loop: do $\cdots$ while (condition)
for	for loop: for (range of values) $\cdots$
$\{\cdots\}$	block of statements
, or . or new-line	statement terminator
/* $\cdots$ */	comment
return	return from this routine

Miscellaneous

■	end of (visible or invisible) proof
□	end of item (like example, definition, assumption)
[H]	hint available in Appendix D

Overview

1.1 Introduction
1.2 Common Cryptographic Primitives
1.3 Public-key Cryptography
1.4 Some Cryptographic Terms
Chapter Summary

1

Aller Anfang ist schwer: All beginnings are difficult.
— German proverb

Defendit numerus: There is safety in numbers.
— Anonymous

The ability to quote is a serviceable substitute for wit.
— W. Somerset Maugham

1.1 Introduction

It is rather difficult to give a precise definition of *cryptography*. Loosely speaking, it is the science (or art or technology) of preventing access to sensitive data by parties who are not authorized to access the data. Secure transmission of messages over a public channel is the first, simplest and oldest example of a cryptographic protocol. For assessing the security of these protocols, one studies their possible weak points, namely the strategies for breaking them. This study is commonly referred to as *cryptanalysis*. And, finally, the study of both cryptography and cryptanalysis is known as *cryptology*.

$$\boxed{\text{Cryptology} = \text{Cryptography} + \text{Cryptanalysis}}$$

The science of cryptology is rather old. It naturally developed as and when human beings felt the need for privacy and secrecy. The rapid deployment of the Internet in the current years demands that we look into this subject with a renewed interest. Newer requirements tailored to Internet applications have started cropping up and as a result newer methods, protocols and algorithms are coming up. The most startling discoveries include that of the key-exchange protocol by Diffie and Hellman in 1976 and that of the RSA cryptosystem by Rivest, Shamir and Adleman in 1978. They opened up a new branch of cryptology, namely *public-key cryptology*. Historically, public-key technology came earlier than the Internet, but it is the latter that makes an extensive use of the former.

This book is an attempt to introduce to the reader the vast and interesting branch of public-key cryptology. One of the most distinguishing features of public-key cryptology is that it involves a reasonable amount of abstract mathematics which often comes in the way of a complete understanding to an uninitiated reader. This book tries to bridge the gap. We develop the required mathematics in necessary and sufficient details.

This chapter is an overview of the topics that the rest of the book deals with. We start with a description of the most common cryptographic protocols. Then we introduce the public-key paradigm and discuss the source of its security. We use certain mathematical terms and notations throughout this chapter. If the reader is not already familiar with these terms, there is nothing to worry about. As we have just claimed, we will introduce the mathematics in the later chapters. The exposition of this chapter is expected to give the reader an overview of the area of public-key cryptography and also the requisite motivation for learning the mathematical tools that follow.

1.2 Common Cryptographic Primitives

As claimed at the outset of this chapter, it is rather difficult to give a precise definition of the term *cryptography*. The best way to understand it is by examples. In this section, we briefly describe the common problems that cryptography deals with.

1.2.1 The Classical Problem: Secure Transmission of Messages

To start with, we introduce the legendary figures of cryptography: Alice, Bob and Carol. Alice wants to send a message to Bob over a public communication channel

like the Internet and wants to ensure that nobody other than Bob can make out the meaning of the message. A third party like Carol, who has access to the communication channel, can intercept the message. But the message should be wrapped or transformed before transmission in such a way that knowledge of some secret piece of information is needed to unwrap or transform back the message. It is Bob who has this information, but not Carol (nor Dorothy nor Emily nor . . .).

It is expedient to point out here that Alice, Bob and Carol need not be human beings. They can stand for organizations (like banks) or, more correctly, for computers or computer programs run by individuals or organizations. It is, therefore, customary to call them *parties*, *entities* or *subjects* instead of persons or characters. In the cryptology jargon, Carol has got several names used interchangeably: *adversary*, *eavesdropper*, *opponent*, *intruder*, *attacker* and *enemy* are the most common ones. When a message transmission like that just mentioned is involved, Alice is called the *sender* and Bob is called the *receiver* of the message.

It is a natural strategy to put the message in a box and lock the box using a *key*, called the *encryption key*. A matching *decryption key* is needed to unlock the box and retrieve the message. The process of putting the message in the box is commonly called *encoding* and that of locking the box is called *encryption*. The reverse processes, namely unlocking the box and taking the message out of the box are respectively called *decryption* and *decoding*. This is precisely the classical *encryption–decryption protocol* of cryptography.[1]

In the world of electronic communication, a message M is usually a bit string, and encoding, encryption, decryption and decoding are well-defined transformations of bit strings. If we denote by f_e the transformation function consisting of encoding and encryption, then we get a new bit string $C = f_e(M, K_e)$, where K_e stands for the encryption key. This bit string C is sent over the communication channel. After Bob receives C, he uses the reverse transformation f_d (decryption followed by decoding) to get the original message M back; that is, $M = f_d(C, K_d)$. Note that the decryption key K_d is needed as an argument to f_d. If Carol does not know this, she cannot compute M. We conventionally call M the *plaintext message* and C the *ciphertext message*.

The encoding and decoding operations do not make use of keys and can be performed by anybody. (It should not be difficult to put a letter in or take a letter out of an unlocked box!) One might then wonder why it is necessary to do these transformations instead of applying the encryption and decryption operations directly on M and C respectively. With whatever we have discussed so far, we cannot give a full answer to this question. For the answer, we will need to wait until we reach the later chapters. We only mention here that the encryption algorithms often require as input some mathematical entities (like integers or elements of a field) which are *logically* not bit strings. But that's not all! As we see later, the additional transformations often add to the security of the protocols. On the other hand, for a general discussion, it is often unnecessary to start from the encoding process and end at the decoding process. As a result, we will assume, unless otherwise stated, that M is the input to the encryption routine and the output of the decryption routine, in which case f_e and f_d stand for the encryption and decryption functions only.

[1] Some people prefer to use the terms *enciphering* and *deciphering* in place of the words *encryption* and *decryption* respectively.

Symmetric-key or secret-key cryptography

In the simplest form of locking mechanism, one has $K_e = K_d$. That is, the same key, called the *symmetric key* or the *secret key*, is used for both encryption and decryption. Common examples of such symmetric-key algorithms include DES (Data Encryption Standard) together with its various modifications like the Triple DES and DES-X, IDEA (International Data Encryption Algorithm), SAFER (Secure And Fast Encryption Routine), FEAL (Fast Encryption Algorithm), Blowfish, RC5 and AES (Advanced Encryption Standard). We will not describe all these algorithms in this book. Interested readers can look at the abundant literature to know more about them.

Asymmetric-key or public-key cryptography

The biggest disadvantage of using a secret-key system is that Alice and Bob must agree upon the key $K_e = K_d$ *secretly*, for example by personal contact or over a secure channel. This is a serious limitation and is not often practical nor even possible. Another drawback of secret-key systems is that every pair of parties needs a key for communication. Thus, if there are n entities communicating over a net, the number of keys would be of the order of n^2. Also, each entity has to *remember* $\mathrm{O}(n)$ keys for communicating with other entities. In practice, however, an entity does not communicate with every other entity on the net. Yet the total number of keys to be remembered by an entity could be quite high.

Both these problems can be avoided by using what is called an *asymmetric-key* or a *public-key* protocol. In such a protocol, each entity decides a *key pair* (K_e, K_d), makes the encryption key K_e public and keeps the decryption key K_d secret. K_e is also called the *public key* and K_d the *private key*. Anybody who wants to send a message to Bob gets Bob's public key, encrypts the message with the key, and sends the ciphertext to Bob. Upon receiving the ciphertext, Bob uses his private key to decrypt the message. One may view such a lock as a self-locking padlock. Anybody can lock a box with a self-locking padlock, but opening it requires a key which only Bob possesses.

The source of security of such a system is based on the difficulty of computing the private key K_d given the public key K_e. It is apparent that K_e and K_d are sort of *inverses* of each other, because the former is used to generate C from M and the latter is used to generate M from C. This is where mathematics comes into the picture. We mention a few possible constructions of key pairs in the next section and the rest of the book deals with an in-depth study of these public-key protocols.

Attractive as it looks, public-key protocols have a serious drawback, namely they are orders of magnitude slower than their secret-key counterparts. This is of concern, if huge amounts of data need to be encrypted and decrypted. This shortcoming can be overcome by using both secret-key and public-key protocols in tandem as follows: Alice generates a secret key (say, for AES), encrypts the message by the secret key and the secret key by the public key of Bob and sends both the encrypted message and the encrypted secret key. Bob first decrypts the encrypted secret key using his private key and uses this decrypted secret key to decrypt the message. Since secret keys are usually short bit strings (most commonly of length 128 bits), the slow performance of the public-key algorithms causes little trouble. But at the same time, Alice and Bob are relieved of having a previous secret meeting or communication for agreeing on the

secret key. Moreover, neither Alice nor Bob needs to remember the secret key. During every session of message transmission, a random secret key can be generated and later destroyed, when the communication is over.

1.2.2 Key Exchange

There is an alternative method by which Alice and Bob can exchange secret information (like AES keys) over a public communication channel. Let us first see how this can be done in the physical lock-and-key scenario. Alice generates a secret, puts it in a box, locks the box with her own key and sends it to Bob. Bob, upon receiving the locked box, adds a second lock to it and sends the doubly locked box back to Alice. Alice then removes her lock and again sends the box to Bob. Finally, Bob uses his key to unlock the box and retrieve the secret. A third party (Carol) that can access the box during the three communications finds it locked by Alice or Bob or both. Since Carol does not possess the keys to these locks, she cannot open the box to discover the secret.

This process can be abstractly described as follows: Alice and Bob first independently generate key pairs (A_{K_e}, A_{K_d}) and (B_{K_e}, B_{K_d}) respectively. Alice then sends A_{K_e} to Bob and Bob sends B_{K_e} to Alice. The private keys A_{K_d} and B_{K_d} are not disclosed. They also agree upon a function g with which Alice computes $g_A = g(A_{K_d}, B_{K_e})$ and Bob computes $g_B = g(B_{K_d}, A_{K_e})$. If $g_A = g_B$, then this common value can be used as a shared secret between Alice and Bob.

Our intruder Carol knows g and taps the values of A_{K_e} and B_{K_e}. So the function g should be such that a knowledge of these values alone does not suffice for the computation of $g_A = g_B$. One of the private keys A_{K_d} or B_{K_d} is needed for the computation. Since (A_{K_e}, A_{K_d}) and (B_{K_e}, B_{K_d}) are key pairs, it is assumed that private keys are difficult to compute from the knowledge of the corresponding public keys.

Such a technique of exchanging secret values over an insecure channel is called a *key-exchange* or a *key-agreement* protocol. It is important to point out here that such a protocol is usually based on the public-key paradigm; that is to say, we do not know secret-key counterparts for a key-exchange protocol. Since a shared secret between the communicating parties is usually short, the low speed of public-key algorithms is really not a concern in this case.

1.2.3 Digital Signatures

A digital signature is yet another application of the public-key paradigm. Suppose Alice wants to sign a message M in such a way that the signature S can be verified by anybody but nobody other than Alice would be able to generate the signature S on the message M. This can be achieved as follows: Alice generates a key pair (K_e, K_d), makes K_e public and keeps K_d secret. She now uses the decryption function f_d to generate the signature, that is, $S = f_d(M, K_d)$. The signature S is then made public. Anybody who has access to Alice's public key K_e applies the reverse transformation f_e to get back the message $M = f_e(S, K_e)$.

If Carol signs the message M with a different key K'_d, then she generates the signature $S' = f_d(M, K'_d)$. Now, since K'_d and K_e are not matching keys, verification using K_e gives $M' = f_e(S', K_e)$, which is different from M. If we assume that M is a

message written in a human-readable language (like English), then M' would generally look like a meaningless sequence of characters which is neither English nor any sensible string to a human reader. So the signature verifier would then immediately conclude that this is a case of forged signature.

Such a scheme of generating digital signatures is called a *signature scheme with message recovery*. It is obvious that this is the same as our encrypt–decrypt scheme with the sequence of encryption and decryption steps reversed. If the message M to be signed is quite long, using this algorithm calls for a large execution time both for signature generation and for verification. It is, therefore, customary to use another variant of signature schemes called *signature schemes with appendix* that we describe now.

Instead of applying the decryption transform directly on M, Alice first computes a *short* representative $H(M)$ of her message M. Her signature now becomes the pair $S = (M, \sigma)$, where $\sigma = f_d(H(M), K_d)$. Typically, a hash function (see Section 1.2.6) is used to compute the representative $H(M)$ from M and is assumed to be a public knowledge. Now anybody can verify the signature by checking if the equality $H(M) = f_e(\sigma, K_e)$ holds. If a key different from K_d is used to generate the signature, one would (in general) get a value $\sigma' \neq \sigma$ and the signature forging will be detected by observing that $H(M) \neq f_e(\sigma', K_e)$.

1.2.4 Entity Authentication

By *entity authentication*, we mean a process in which one entity called the *claimant* proves its identity to another entity called the *prover*. Entity-authentication techniques, thus, tend to prevent impersonation of an entity by an intruder. Both secret-key and public-key techniques are used for entity-authentication schemes.

The simplest example of an entity-authentication scheme is the use of *passwords*, as in a computer where a user (the claimant) tries to gain access to some resources in a computer (the prover) by proving its identity using a password. Password schemes are mostly based on secret-key techniques. For example, the UNIX password system is based on encrypting the zero message (a string of 64 zero bits) using a repeated application of a variant of the DES algorithm with 64 bits of the user input (the password) as the key. Password-based authentication schemes are fixed and time-invariant and are often called *weak authentication* schemes.

We see applications of public-key techniques in *challenge–response authentication schemes* (also called *strong authentication schemes*). Assume that an entity, Alice, wants to prove her identity to another entity, Bob. Alice generates a key pair (K_e, K_d), makes K_e public and keeps K_d secret. Now, Bob chooses a random message M, encrypts M using Alice's public key—that is, computes $C = f_e(M, K_e)$—and sends C to Alice. Alice, upon reception of C, decrypts it using her private key K_d; that is, she regenerates $M = f_d(C, K_d)$ and sends M to Bob. Bob compares this value of M with the one he generated, and if a match occurs, Bob becomes sure that the entity who is claiming to be Alice possesses the knowledge of Alice's private key. If Carol uses any private key other than K_d for the decryption, she gets a message M' different from M and thereby cannot prove to Bob her identity as Alice. This is how this scheme prevents impersonation of Alice by Carol.

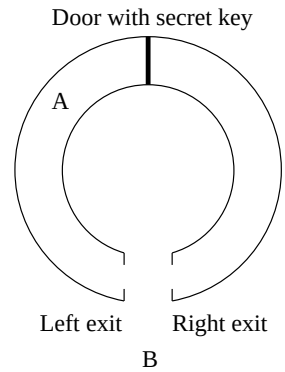


Figure 1.1 Zero-knowledge proofs

Entity authentication is often carried out using another interesting technique called *zero-knowledge proof*. In such a protocol, the prover (or any third party listening to the conversation) gains *no knowledge* regarding the secret possessed by the claimant, but develops the desired confidence regarding the claim by the claimant of the *possession* of the secret. We provide here an informal example explaining zero-knowledge proofs.

Let us think of a circular cave as shown in Figure 1.1. The cave has two exits, left and right, denoted by L and R respectively. The cave also has a door inside it, which is invisible outside the cave. Alice (A) wants to prove to Bob (B) that she possesses a key to this door without showing him the key or the process of unlocking the door with the key. Bob stations himself somewhere outside the exits of the cave. Alice enters the cave and randomly chooses the left or right wing of the cave (and goes there). She does not disclose this choice to Bob, because Bob is not allowed to know the session secrets too. Once Alice is placed in the cave, Bob makes a random choice from L and R and asks Alice (using cell phones or by shouting loudly) to come out of the cave via that chosen exit. Suppose Bob challenges Alice to use L. If Alice is in the left wing, she can come out of the cave using L. If Alice is in the right wing, she must use her secret key to open the central door to come to the left wing and then go out using exit L. If Alice does *not* possess the secret key, she can succeed in obeying Bob's directive with a probability of half. If this procedure is repeated t times, then the probability that Alice succeeds on all occasions without possessing the secret key is $(1/2)^t = 1/2^t$. By choosing t appropriately, Bob can make the probability of accepting a false claim arbitrarily small. For example, if $t = 20$, then the chance is less than one in a million that Alice can establish a false claim.

Thus, if Alice succeeds every time, Bob gains the desired confidence that Alice actually possesses the secret. However, during this entire process, Bob can obtain no information regarding Alice's secrets (the key and the choices of wings). Another important aspect of this interaction is that Alice has no way of predicting Bob's questions, preventing impostors (of Alice) from fooling Bob.

1.2.5 Secret Sharing

Suppose that a secret piece of information is to be distributed among n entities in such a way that $n-1$ (or fewer) entities are unable to construct the secret. All of the n entities must participate to reveal the secret. As usual, let us assume that the secret is an l-bit string. A simple strategy would be to break the string into n parts and provide each entity with a part. This method is, however, not really attractive, because it gives partial information about the secret. Thus, for example, if a 256-bit long bit string is to be distributed equally among 16 entities, any 15 of them working together can reconstruct the secret by trying only $2^{16} = 65536$ possibilities for the unknown 16 bits.

We now describe an alternative strategy that does not suffer from this drawback. Once again, we break the secret string into n parts and consider the parts as integers $a_0, \ldots, a_{n-1}$. We construct the polynomial $f(x) = x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$ and give the integers $f(1), f(2), \ldots, f(n)$ to the entities. When all of the entities cooperate, the linear system of equations $f(i) = i^n + a_{n-1}i^{n-1} + \cdots + a_1 i + a_0$, $1 \leqslant i \leqslant n$, can be solved to find out the unknown coefficients $a_0, \ldots, a_{n-1}$ which, in turn, reveal the secret. On the other hand, if $n-1$ or less entities cooperate, they get an underspecified system of equations in n unknowns, from which the actual solution is not readily available.

The secret-sharing problem can be generalized in the following way: to distribute a secret among n parties in such a way that any m or more of the parties can reconstruct the secret (for some $m \leqslant n$), whereas any $m-1$ or less parties cannot do the same. A polynomial of degree m as in the above example readily adapts to this generalized situation.

1.2.6 Hashing

A function which converts bit strings of arbitrary lengths to bit strings of a fixed (finite) length is called a *hash function*. Hash functions play a crucial role in cryptography. We have already seen an application of it for designing a digital signature scheme with appendix. If H is a hash function, a pair of input values (strings) x_1 and x_2 for which $H(x_1) = H(x_2)$ is called a *collision* for H. For any hash function H, collisions must exist, since H is a map from an infinite set to a finite set. However, for cryptographic purposes we want that collisions should be difficult to obtain. More specifically, a cryptographic hash function H should satisfy the following desirable properties:

First pre-image resistance Except for a *small* set of hash values y it should be difficult to find an input x with $H(x) = y$. We exclude a small set of values, because an adversary might prepare (and maintain) a list of pairs $(x, H(x))$ for certain values of x of her choice. If the given value of y is the second coordinate of one pair in her list, she can produce the corresponding input value x easily.

Second pre-image resistance Given a pair $(x, H(x))$, it should be difficult to find an input x' different from x with $H(x) = H(x')$.

Collision resistance It should be difficult to find two different input strings x, x' with $H(x) = H(x')$.

Hash functions are also called *message digests* and can be used with a secret key. Popular examples of unkeyed hash functions are SHA-1, MD5 and MD2, whereas those for keyed hash functions include HMAC and CBCMAC.

1.2.7 Certification

So far we have seen several protocols which are based on the use of public keys of remote entities, but have never questioned the authenticity of public keys. In other words, it is necessary to ascertain that a public key is really owned by a remote entity. *Public-key certificates* are used to that effect. These are data structures that bind public-key values to entities. This binding is achieved by having a trusted *certification authority* digitally sign each certificate.

Typically a certificate is issued for a period of validity. However, it is possible that a certificate becomes invalid before its date of expiry for several reasons, like possible or suspected compromise of the private key. Under such circumstances it is necessary that the certification authority *revokes* the certificate and maintains a list called *certificate revocation list* (CRL) of revoked certificates. When Alice verifies the authenticity of Bob's public-key certificate by verifying the digital signature of the authority and does not find the certificate in the CRL, she gains the desired confidence in using Bob's public key.

The X 5.09 *public-key infrastructure* specifies Internet standards for certificates and CRLs.

1.3 Public-key Cryptography

In this section, we give a short introduction to the realization of public-key cryptosystems. More specifically, we list some of the computationally intensive mathematical problems and describe how the (apparent) intractability of these problems can be used for designing key pairs. We use some mathematical terms that we will introduce later in this book.

1.3.1 The Mathematical Problems

The security of the public-key cryptosystems is based on the *presumed* difficulty of solving certain mathematical problems.

The integer factorization problem (IFP) Given the product $n = pq$ of two distinct prime integers p and q, find p and q.

The discrete logarithm problem (DLP) Let G be a finite cyclic (multiplicatively written) group with cardinality n and a generator g. Given an element $a \in G$, find an integer x (or *the* integer x with $0 \leqslant x \leqslant n-1$) such that $a = g^x$ in G. Three different types of groups are commonly used for cryptographic applications: the multiplicative group of a finite field, the group of rational points on an elliptic curve over a finite field and the Jacobian of a hyperelliptic curve over a finite field. By an abuse of notation,

we often denote the DLP over finite fields as simply DLP, whereas the DLP in elliptic curves and hyper-elliptic curves is referred to as the *elliptic curve discrete logarithm problem* (ECDLP) and the *hyperelliptic curve discrete logarithm problem* (HECDLP).

The Diffie–Hellman problem (DHP) Let G and g be as above. Given elements g^a and g^b of G, compute the element g^{ab}. As in the case of the DLP, the DHP can be applied to the multiplicative group of finite fields, the group of rational points on an elliptic curve and the Jacobian of a hyperelliptic curve.

We show in the next section how (the intractability of) these problems can be exploited to create key pairs for various cryptosystems. These computational problems are termed *difficult*, *intractable*, *infeasible* or *intensive* in the sense that there are no *known* algorithms to solve these problems in time polynomially bounded by the input size. The best-known algorithms are *subexponential* or even fully *exponential* in some cases. This means that if the input size is chosen to be sufficiently large, then it is infeasible to compute the private key from a knowledge of the public key in a reasonable amount of time. This, in turn, implies (not provably, but as the current state of the art stands) that encryption or signature verification can be done rather quickly (in polynomial time), but the converse process of decryption or signature generation cannot be done in feasible time, unless one knows the private key. As a result, encryption (or signature verification) is called a *trapdoor one-way function*, that is, a function which is easy to compute but for which the inverse is computationally infeasible, unless some additional information (the trapdoor) is available.

It is, however, not known that these problems are really computationally infeasible, that is, there is no proof of the fact that these problems cannot be solved in polynomial time. As a result, the public-key cryptographic systems based on these problems are *not provably secure*.

1.3.2 Realization of Key Pairs

In RSA and similar cryptosystems, one generates two (distinct) suitably large primes p and q and computes the product $n = pq$. Then $\phi(n) = (p-1)(q-1)$, where ϕ denotes Euler's totient function. One then chooses a random integer e with $\gcd(e, \phi(n)) = 1$. There exists an integer d such that $ed \equiv 1 \pmod{\phi(n)}$. The integer e is used as the public key, whereas the integer d is used as the private key.

If the IFP can be solved fast, one can also compute $\phi(n)$ easily, and subsequently d can be computed from e using the (polynomial-time) extended GCD algorithm. This is why[2] we say that the RSA cryptosystem derives its security from the intractability of the IFP.

In order to see how RSA encryption and decryption work, let the plaintext message be encoded as an integer m with $2 \leqslant m < n$. The ciphertext message is generated (as an integer) as $c = m^e \pmod{n}$. Decryption is analogous, that is, $m = c^d \pmod{n}$. The correctness of the algorithm follows from the fact that $ed \equiv 1 \pmod{\phi(n)}$. It is, however, not proved that one *has to* know d or $\phi(n)$ or the factorization of n in order to decrypt an RSA-encrypted message. But at present no better methods are known.

[2]The problem of factoring $n = pq$ is polynomial-time equivalent to computing $\phi(n) = (p-1)(q-1)$.

Let us now consider the discrete logarithm problem. Let G be a finite cyclic multiplicative group (as those mentioned above) where it is easy to multiply two elements, but where it is difficult to compute discrete logarithms. Let g be a generator of G. In order to set up a random key pair over such a group, one chooses the private key as a random integer d, $2 \leqslant d < n$, where n is the cardinality of G. The public key e is then computed as an element of G as $e = g^d$.

Applications of encryption–decryption schemes based on the key pair (g^d, d) are given in Chapter 5. Now, we only remark that many such schemes (like the ElGamal scheme) derive their security from the DHP instead of the DLP, whereas the other schemes (like the Nyberg–Rueppel scheme) do so from the DLP. It is assumed that these two problems are computationally equivalent (at least for the groups of our interest). Obviously, if one assumes availability of a solution of the DLP, one has a solution for the DHP too ($g^{ab} = (g^a)^b$). The reverse implication is not clear.

1.3.3 Public-key Cryptanalysis

As we pointed out earlier, (most of) the public-key cryptosystems are not provably secure in the sense that they are based on the *apparent* difficulty of solving certain computational problems. It is expedient to know how difficult these problems are. No non-trivial complexity–theoretic statements are available for these problems, and as such it is worthwhile to study the algorithms known till date to solve these problems. Unfortunately, however, many of the algorithms of this kind are often much more complicated than the algorithms for building the corresponding cryptographic systems. One needs to acquire more mathematical machinery in order to understand (and augment) these cryptanalytic algorithms. We devote Chapter 4 to a detailed discussion on these algorithms.

In specific situations, one need not always use these computationally intensive algorithms. Access to a party's decryption equipment may allow an adversary to gain partial or complete information about the private key by *watching* a decryption process. For example, an adversary (say, the superuser) might have the capability to read the contents of the memory holding a private key during some decryption process. For another possibility, think of RSA decryption which involves a modular exponentiation. If the standard square-and-multiply algorithm (Algorithm 3.9) is used for this purpose and the adversary can tap some hardware details (like machine cycles or power fluctuations) during a decryption process, she can guess a significant number of the bits in the private key. Such attacks, often called *side-channel attacks*, are particularly relevant for cryptographic applications based on smart cards.

A cryptographic system is (believed to be) strong if and only if there are no good known mechanisms to break it. It is, therefore, for the sake of security that we must study cryptanalysis. Cryptography and cryptanalysis are deeply intertwined and a complete study of one must involve the other.

1.4 Some Cryptographic Terms

In cryptology, there are different models of attacks or attackers.

1.4.1 Models of Attacks

So far we have assumed that an adversary can *only* read messages during transmission over a channel. Such an adversary is called a *passive adversary*. An *active adversary*, on the other hand, can mutilate or delete messages during transmission and/or generate false messages. An attack mounted by an active (resp.[3] a passive) adversary is called an *active* (resp. a *passive*) *attack*. In this book, we will mostly concentrate on passive attacks.

1.4.2 Models of Passive Attacks

A two-party communication involves transmission of ciphertext messages over a communication channel. A passive attacker can read these ciphertext messages. In practice, however, an attacker might have more control over the choice of ciphertext and/or plaintext messages. Based on these capabilities of the attacker we have the following types of attacks.

Ciphertext-only attack This is the weakest model of the adversary. Here the attacker has absolutely no choices on the ciphertext messages that flow in the channel and also on the corresponding plaintext messages. Using only these ciphertext messages the attacker has to obtain a private key and/or a plaintext message corresponding to a new ciphertext message.

Known-pair attack In this kind of attack (also called *known-plaintext* or *known-ciphertext attack*), the attacker uses her knowledge of some plaintext–ciphertext pairs. If many such pairs are available to the attacker, she can use these pairs to deduce a pattern based on which she can subsequently gain some information on a new plaintext for which the ciphertext is available. In a public-key scheme, the adversary can generate as many such pairs as she wants, because in order to generate such a pair it is sufficient to have a knowledge of the receiver's public key. Thus a public-key encryption scheme must provide sufficient security against known plaintext attacks.

Chosen-plaintext attack In this kind of attack, the attacker knows some plaintext–ciphertext pairs in which the plaintexts are *chosen* by the attacker. As discussed earlier, such an attack is easily mountable for a public-key encryption scheme.

Adaptive chosen-plaintext attack This is similar to the chosen-plaintext attack with the additional possibility that the attacker chooses the plaintexts in the known plaintext–ciphertext pairs sequentially and *adaptively* based on the knowledge of the previous pairs. This kind of attack can be easily mounted on public-key encryption systems.

Chosen-ciphertext attack The attacker has knowledge of some plaintext–ciphertext pairs in which the ciphertexts are *chosen* by the attacker. Such an attack is not directly mountable on a public-key scheme, since obtaining a plaintext from a chosen ciphertext requires knowledge of the private key. However, if the attacker has access to the receiver's decryption equipment, the machine can divulge the plaintexts corresponding to the ciphertexts that the attacker supplies to the machine. In this context, we assume that the machine does not reveal the private key itself, that is, it has the key

[3] Throughout the book, resp. stands for respectively.

stored secretly somewhere in its hardware which the attacker cannot directly access. However, the attacker can run the machine to know the plaintexts corresponding to the ciphertexts of her choice. Later (when the attacker no longer has access to the decryption equipment) the known pairs may be exploited to obtain information about the plaintext corresponding to a new ciphertext.

Adaptive chosen-ciphertext attack This is similar to the chosen-ciphertext attack with the additional possibility that the attacker chooses the ciphertexts in the known pairs sequentially and *adaptively* based on her knowledge of the previously generated plaintext–ciphertext pairs. This attack is mountable in a scenario described in connection with chosen-ciphertext attacks.

For a digital signature scheme, there are equivalent names for these types of attacks. The attacker is assumed to have access to the public key of the signer, because this key is used for signature verification. An attempt to forge signatures based only on the knowledge of this verification key is called a *key-only attack*. The adversary may additionally possess knowledge of some message–signature pairs. An attack based on this knowledge is called a *known-pair* or *known-message* or *known-signature attack*. If the messages are *chosen* by the adversary, we call the attack a *chosen-message attack*. If the adversary generates the sequence of messages in a chosen-message attack adaptively (based on the previously generated message–signature pairs), we have an *adaptive chosen-message attack*. An (adaptive or non-adaptive) chosen-message attack can be mounted, if the attacker gains access to the signer's signature generation equipment, or if the signer is willing to sign arbitrary messages provided by the adversary.

The attacker can choose some signatures and generate the corresponding messages by encrypting them with the signer's public key. The private-key operation on these messages generates the signatures chosen by the attacker. This gives *chosen-signature* and *adaptive chosen-signature attacks* on a digital signature scheme. Now the adversary cannot directly control the messages to sign. On the other hand, such an attack is easily mountable, because it utilizes only some public knowledge (the signer's public key). Indeed, one may treat chosen-signature attacks as variants of key-only attacks.

1.4.3 Public Versus Private Algorithms

So far, we have assumed that all the parties connected to a network *know* the algorithms used in a cryptographic scheme. The security of the scheme is based on the difficulty of obtaining some secret information (the secret or private key).

It, however, remains possible that two parties communicate using an algorithm unknown to other entities. Top-secret communications (for example, during wars or diplomatic transactions) often use private cryptographic algorithms. In this book, we will not deal with such techniques. Our attention is focused mostly on Internet applications in which public knowledge of the algorithms is of paramount importance (for the sake of universal applicability and convenience).

In short, this book is going to deal with a world in which only public public-key algorithms are deployed and in which adversaries are usually passive. A restricted model of the world though it may be, it is general and useful enough to concentrate on. Let us begin our journey!

Chapter Summary

This chapter provides an overview of the problems that *cryptology* deals with. The first and oldest cryptographic primitive is *encryption* for secure transmission of messages. Some other primitives are *key exchange*, *digital signature*, *authentication*, *secret sharing*, *hashing*, and *digital certificates*. We then highlight the difference between *symmetric* (secret-key) and *asymmetric* (*public-key*) *cryptography*. The relevance of some computationally intractable mathematical problems in public-key cryptography is discussed next, and the working of a prototype public-key cryptosystem (RSA) is explained. We finally discuss different models of attacks on cryptosystems.

Not uncommonly, some people think that cryptology also deals with intrusion, viruses, and Trojan horses. We emphasize that this is never the case. *Data and network security* is the branch that deals with these topics. Cryptography is also a part of this branch, but not conversely. Imagine that your house is to be secured against theft. First, you need a good lock—that is, cryptography. However, a lock has nothing to prevent a thief from entering the house after breaking the window panes. A bad butler who leaks secret information of the house to the outside world also does not come under the jurisdiction of the lock. Securing your house requires adopting sufficient guards against all these possibilities of theft. In this book, we will study only the technology of manufacturing and breaking locks.

Mathematical Concepts

2

2.1 Introduction
2.2 Sets, Relations and Functions
2.3 Groups
2.4 Rings
2.5 Integers
2.6 Polynomials
2.7 Vector Spaces and Modules
2.8 Fields
2.9 Finite Fields
2.10 Affine and Projective Curves
2.11 Elliptic Curves
2.12 Hyperelliptic Curves
2.13 Number Fields
2.14 p-adic Numbers
2.15 Statistical Methods
Chapter Summary
Sugestions for Further Reading

Young man, in mathematics you don't understand things, you just get used to them.
— John von Neumann

Mathematics contains much that will neither hurt one if one does not know it nor help one if one does know it.
— J. B. Mencken

Mathematics is the Queen of Science but she isn't very pure; she keeps having babies by handsome young upstarts and various frog princes.
— Donald Kingsbury

2.1 Introduction

In this chapter, we introduce the basic mathematical concepts that one should know in order to understand the public-key cryptographic protocols and the corresponding cryptanalytic algorithms described in the later chapters. If the reader is already familiar with these concepts, she may quickly browse through the chapter in order to know about our notations and conventions.

This chapter is meant for cryptology students and as such does not describe the mathematical topics in their full generality. It is our intention only to state (and, if possible, prove) the relevant results that would be useful for the rest of the book. For further study, we urge the reader to consult the books suggested at the end of this chapter.

2.2 Sets, Relations and Functions

Sets are absolutely basic entities used throughout the present-day study of mathematics. Unfortunately, however, we cannot define sets. Loosely speaking, *a set is an (unordered) collection of objects*. But we run into difficulty with this definition for collections that are *too big*. Of course, infinite sets like the set of all integers or real numbers are not too big. However, a collection of all sets is too big to be called a set. (Also see Exercise 2.6.) It is, therefore, customary to have an *axiomatic definition* of sets. That is to say, a collection qualifies to be a set if it satisfies certain axioms. We do not go into the details of this axiomatic definition, but tell the axioms as *properties* of sets. Luckily enough, we won't have a chance in the rest of this book to deal with collections that are not sets. So the reader can, for the time being, have faith in the above (wrong) identification of a set as a collection.

An object in a set is commonly called an *element* of A. By the notation $a \in A$, we mean that a is an element of the set A. Often a set A can be represented explicitly by writing down its elements within curly brackets or braces. For example, $A = \{2, 3, 5, 7\}$ denotes the set consisting of the elements $2, 3, 5, 7$ which are incidentally all the (positive) prime numbers less than 10. We often use the ellipsis sign (. . .) to denote an infinite (or even a finite) set. For example, $\mathbb{P} = \{2, 3, 5, 7, \ldots\}$ would denote the set of all (positive) prime numbers. (We prove later that $\mathbb{P}$ is an infinite set.) Alternatively, we often describe a set by mentioning the properties of the elements of the set. For example, the set $\mathbb{P}$ can also be described as $\mathbb{P} = \{a \mid a \text{ is a (positive) prime number}\}$.

Some frequently occurring sets are denoted by special symbols. We list a few of them here.

$\mathbb{N}$	The set of all natural numbers, that is, $\{1, 2, 3, \ldots\}$
$\mathbb{N}_0$	The set of all non-negative integers, that is, $\{0, 1, 2, \ldots\}$
$\mathbb{Z}$	The set of all integers, that is, $\{\ldots, -2, -1, 0, 1, 2, \ldots\}$
$\mathbb{P}$	The set of all (positive) prime numbers, that is, $\{2, 3, 5, 7, \ldots\}$
$\mathbb{Q}$	The set of all rational numbers, that is, $\{a/b \mid a \in \mathbb{Z}, b \in \mathbb{N}\}$
$\mathbb{Q}^*$	The set of all non-zero rational numbers
$\mathbb{R}$	The set of all real numbers
$\mathbb{R}^*$	The set of all non-zero real numbers

$\mathbb{C}$	The set of all complex numbers
$\mathbb{C}^*$	The set of all non-zero complex numbers
$\emptyset$	The empty set

The *cardinality* of a set A is the number of elements in A. We use the symbol $\#A$ to denote the cardinality of A. If $\#A$ is finite, we call A a *finite set*. Otherwise A is said to be *infinite*. The empty set has cardinality zero.

2.2.1 Set Operations

Let A and B be two sets. We say that A is a *subset* of B and denote this as $A \subseteq B$, if all elements of A are in B. Two sets A and B are equal (that is, $A = B$) if and only if $A \subseteq B$ and $B \subseteq A$. A is said to be a *proper subset* of B (denoted $A \subsetneqq B$), if $A \subseteq B$ and $A \neq B$ (that is, $B \not\subseteq A$).

The *union* of A and B is the set whose elements are either in A or in B (or both). This set is denoted by $A \cup B$. The *intersection* of A and B is the set consisting of elements that are common to A and B. The intersection of A and B is denoted by $A \cap B$. If $A \cap B = \emptyset$, then we say that A and B are *disjoint*. In that case, the union $A \cup B$ is also called a *disjoint union* and is referred to as by $A \uplus B$. (For a generalization, see Exercise 2.7.) The *difference* of A and B, denoted $A \setminus B$, is the set whose elements are in A but not in B. If A is understood from the context and $B \subseteq A$, then we denote $A \setminus B$ by $\bar{B}$ and refer to $\bar{B}$ as the *complement* of B (in A). The *product* $A \times B$ of two sets A and B is the set of all ordered pairs (a, b) where $a \in A$ and $b \in B$.

The notion of union, intersection and product of sets can be readily extended to an arbitrary family of sets. Let A_i, $i \in I$, be a family of sets indexed by I. In this case, we denote the union and intersection of A_i, $i \in I$, by $\bigcup_{i \in I} A_i$ and $\bigcap_{i \in I} A_i$ respectively. The product of A_i, $i \in I$, is denoted by $\prod_{i \in I} A_i$. When $A_i = A$ for all $i \in I$, we denote the product also as A^I. If, in addition, I is a finite set of cardinality n, then the product A^I is also written as A^n.

2.2.2 Relations

A *relation* ρ on a set A is a subset of $A \times A$. For $(a, b) \in \rho$, we usually say $a \,\rho\, b$ implying that a is related by ρ to b. Common examples are the standard relations $=$, $\neq$, $\leqslant$, $<$, $\geqslant$, $>$ on $\mathbb{Z}$ (or $\mathbb{Q}$ or $\mathbb{R}$).

A relation ρ on a set A is called *reflexive*, if $a \,\rho\, a$ for all $a \in A$. For example, $=$, $\leqslant$ and $\geqslant$ are reflexive relations on $\mathbb{Z}$, but the relations $\neq$, $<$, $>$ are not.

A relation ρ on A is called *symmetric*, if $a \,\rho\, b$ implies $b \,\rho\, a$. On the other hand, ρ is called *anti-symmetric* if $a \,\rho\, b$ and $b \,\rho\, a$ imply $a = b$. For example, $=$ is symmetric and anti-symmetric, $<$, $\leqslant$, $>$ and $\geqslant$ are anti-symmetric but not symmetric, $\neq$ is symmetric but not anti-symmetric.

A relation ρ on A is called *transitive* if $a \,\rho\, b$ and $b \,\rho\, c$ imply $a \,\rho\, c$, For example, $=$, $<$, $\leqslant$, $>$, $\geqslant$ are all transitive, but $\neq$ is not transitive.

An *equivalence relation* is one which is reflexive, symmetric and transitive. For example, $=$ is an equivalence relation on $\mathbb{Z}$, but neither of the other relations mentioned above ($\neq$, $<$, $\geqslant$ and so on) is an equivalence relation on $\mathbb{Z}$.

A *partition* of a set A is a collection of pairwise disjoint subsets A_i, $i \in I$, of A, such that $A = \biguplus_{i \in I} A_i$, that is, A is the union of A_i, $i \in I$, and for $i, j \in I$, $i \neq j$, $A_i \cap A_j = \emptyset$. The following theorem establishes an important connection between equivalence relations and partitions.

Theorem 2.1 An equivalence relation on a set A produces a partition of A. Conversely, every partition of a set A corresponds to an equivalence relation on A.

Proof Let ρ be an equivalence relation on a set A. For $a \in A$, let us denote $[a] := \{b \in A \mid a \,\rho\, b\}$. Clearly, $A = \bigcup_{a \in A} [a]$, since $a \in [a]$ (by reflexivity). Now we show that for $a, b \in A$, either $[a] = [b]$ or $[a] \cap [b] = \emptyset$. Assume that $[a] \cap [b] \neq \emptyset$. Choose $c \in [a]$. By construction, $a \,\rho\, c$. Now choose $d \in [a] \cap [b]$. Then $a \,\rho\, d$ and $b \,\rho\, d$. By symmetry, $d \,\rho\, b$, so that by transitivity $a \,\rho\, b$, that is, $b \,\rho\, a$. But $a \,\rho\, c$. Hence, once again by transitivity, $b \,\rho\, c$, that is, $c \in [b]$. Thus $[a] \subseteq [b]$. Similarly $[b] \subseteq [a]$.

Conversely, let A_i, $i \in I$, be a partition of A. Define a relation ρ on A such that $a \,\rho\, b$ if and only if a and b are in the same subset A_i for some i. It is easy to see that ρ is an equivalence relation on A. ■

The subset $[a]$ of A defined in the proof of the above theorem is called the *equivalence class* of a with respect to the equivalence relation ρ.

An anti-symmetric and transitive relation is called a *partial order* (or simply an *order*). All of the relations $=, \leqslant, <, \geqslant, >$ are partial orders on $\mathbb{Z}$ (but $\neq$ is not). A partial order ρ on A is called a *total order* or a *linear order* or a *simple order*, if for every $a, b \in A$, $a \neq b$, either $a \,\rho\, b$ or $b \,\rho\, a$. For example, if we take $A = \{1, 2, 3\}$ and the relation $\rho = \{(1, 2), (1, 3)\}$, then ρ is a partial order but not a total order (because it does not specify a relation between 2 and 3). On the other hand, $\rho' = \{(1, 2), (1, 3), (2, 3)\}$ is a total order. A set with a partial (resp. total) order is often called a *partially ordered* (resp. *totally ordered* or *linearly ordered* or *simply ordered*) *set*.

2.2.3 Functions

Let A and B two sets (not necessarily distinct). A *function* or a *map* f from A to B, denoted $f : A \to B$, assigns to each $a \in A$ some element $b \in B$. In this case, we write $b = f(a)$ or f maps $a \mapsto b$ and say that b is the *image* of a (under f). For example, if $A = B = \mathbb{R}$, then the assignment $a \mapsto a^2$ is a function. On the other hand, the assignment $a \mapsto \sqrt{a}$ (the non-negative square root) is not a function, because it is not defined for negative values of a. However, if $A = \mathbb{R}$ and $B = \mathbb{C}$, then the assignment $a \mapsto \sqrt{a}$ (with non-negative real and imaginary parts) is a function.

The function $f : A \to A$ assigning $a \mapsto a$ for all $a \in A$ is called the *identity map* on A and is usually denoted by id_A. On the other hand, if $f : A \to B$ maps all the elements of A to a *fixed* element of B, then f is said to be a *constant function*. A function which is not constant is called a *non-constant function*.

A function $f : A \to B$ that maps different elements of A to different elements of B is called *injective* or *one-one*. In other words, we call f to be injective if and only if $f(a) = f(a')$ implies $a = a'$. The function $f : \mathbb{R} \to \mathbb{R}$ given by $a \mapsto a^2$ is not injective, since $f(-a) = f(a)$ for all $a \in \mathbb{R}$. On the other hand, the function $f : \mathbb{R} \to \mathbb{R}$ given by $a \mapsto 2a$ is injective. An injective map $f : A \to B$ is sometimes denoted by the special symbol $f : A \hookrightarrow B$.

The *image* of a function $f : A \to B$ is defined to be the following subset of B: $\{b \in B \mid \exists\, a \in A \text{ with } b = f(a)\}$. It is denoted by $f(A)$ or by $\operatorname{Im} f$. The function f is said to be *surjective* or *onto* or a *surjection*, if $\operatorname{Im} f = B$, that is, every element b of B has at least one *preimage* $a \in A$ (which means $f(a) = b$). As an example, the function $f : \mathbb{Z} \to \mathbb{Z}$ given by $a \mapsto a/2$ (if a is even) and by $a \mapsto (a-1)/2$ (if a is odd) is surjective, whereas the function $f : \mathbb{Z} \to \mathbb{Z}$ that maps $a \to |a|$ (the absolute value) is not surjective. A surjective map $f : A \to B$ is sometimes denoted by the special symbol $f : A \twoheadrightarrow B$.

A map $f : A \to B$ is called *bijective* or a *bijection*, if it is both injective and surjective. For example, the identity map on a set is bijective. Another example of a bijective function is $f : \mathbb{N} \to \mathbb{P}$ that maps a to the ath prime.

Let $f : A \to B$ and $g : B \to C$ be functions. The *composition* of f and g is the function from A to C that takes $a \mapsto g(f(a))$. It is denoted by $g \circ f$, that is, $(g \circ f)(a) = g(f(a))$. Note that in the notation $g \circ f$ one applies f first and then g. The notion of composition of functions can be extended to more than two functions. In particular, if $f : A \to B$, $g : B \to C$ and $h : C \to D$ are functions, then $(h \circ g) \circ f$ and $h \circ (g \circ f)$ are the same function from A to D, so that we can unambiguously write this as $h \circ g \circ f$.

2.2.4 The Axioms of Mathematics

The study of mathematics is based on certain axioms. We state four of these axioms. It is not possible to *prove* the axioms independently, but it can be shown that they are equivalent in the sense that each of them can be proved, if any of the others is assumed to be true.

Let A be a partially ordered set under the relation $\preceq$. An element $a \in A$ is called *maximal* (resp. *minimal*), if there is no element $b \in A$, $b \neq a$, that satisfies $a \preceq b$ (resp. $b \preceq a$). Let B be a non-empty subset of A. Then an *upper bound* (resp. a *lower bound*) for B is an element $a \in A$ such that $b \preceq a$ (resp. $a \preceq b$) for all $b \in B$. If an upper bound (resp. a lower bound) a of B is an element of B, then a is called a *last element* or a *largest element* or a *maximum element* (resp. a *first element* or a *least element* or a *smallest element* or a *minimum element*) of B. By antisymmetry, it follows that a first (resp. last) element of B, if existent, is unique. A *chain* of A is a totally ordered (under $\preceq$) subset of A.

Consider the sets $\mathbb{N}$, $\mathbb{Z}$ and $\mathbb{R}$ with the natural order $\leqslant$. Neither of these sets contains a maximal element. $\mathbb{N}$ contains a minimal element 1, but $\mathbb{Z}$ and $\mathbb{R}$ do not contain minimal elements. The subset $\mathbb{N}_{\text{even}} \subseteq \mathbb{N}$ of even natural numbers has two lower bounds, namely 1 and 2, of which 2 is the first element of $\mathbb{N}_{\text{even}}$.

A totally ordered set A is said to be *well ordered* (and the relation is called a *well order*), if every non-empty subset B of A contains a first element.

Axiom 2.1 **Zermalo's well-ordering principle** Every set A can be well ordered, that is, there is a relation $\preceq$ which well orders A. □

The set $\mathbb{N}$ is well-ordered under the natural relation $\leqslant$. The set $\mathbb{Z}$ can be well ordered by the relation $\preceq$ defined as $0 \preceq 1 \preceq -1 \preceq 2 \preceq -2 \preceq \cdots$. A well ordering of $\mathbb{R}$ is not known.

Axiom 2.2 **Zorn's lemma** Let A be a partially ordered set. If every chain of A has an upper bound (in A), then A has at least one maximal element. □

To illustrate Zorn's lemma, consider any non-empty set A and define $\mathcal{P}(A)$ to be the set of all subsets of A. $\mathcal{P}(A)$ is called the *power set* of A and is partially ordered under containment $\subseteq$. A chain of $\mathcal{P}(A)$ is a set $\{A_i \mid i \in I\}$ of subsets of A such that for all $i, j \in I$ either $A_i \subseteq A_j$ or $A_j \subseteq A_i$. Clearly, the union $\bigcup_{i\in I} A_i$ is an upper bound of the chain. Then Zorn's lemma guarantees that $\mathcal{P}(A)$ has at least one maximal element. In this case, the maximal element, namely A, is unique. If A is finite, then for the set $\mathcal{P}(A) \setminus \{A\}$ of all proper subsets of A, a maximal element (under the partial order $\subseteq$) exists by Zorn's lemma, but is not unique, if $\#A > 1$.

Axiom 2.3 **Hausdorff's maximal principle** Let $\preceq$ be a partial order on a set A. Then there is a maximal chain B of A, that is, if C is any chain with $B \subseteq C \subseteq A$, then $C = B$. □

Finally, let A be a set and $\mathcal{P}'(A) := \mathcal{P}(A) \setminus \{\emptyset\}$, that is, $\mathcal{P}'(A)$ is the set of all non-empty subsets of A. A *choice function* of A is a function $f : \mathcal{P}'(A) \to A$ such that for every $\emptyset \neq B \subseteq A$ we have $f(B) \in B$.

Axiom 2.4 **Axiom of choice** Every set has a choice function. □

EXERCISE SET 2.2

2.1 **(a)** Let $G = (V, E)$ be an undirected graph. Define a relation ρ on the vertex set V of G by: $u \,\rho\, v$ if and only if there is a path from u to v. Show that ρ is an equivalence relation on V. What are the equivalence classes for this relation?
(b) Let $G = (V, E)$ be a directed acyclic graph. Define the relation ρ on V as in (a). Show that ρ is a partial order on V. When is ρ a total order?

2.2 Let $f : A \to B$ and $g : B \to A$ be functions. Show that if $f \circ g = \mathrm{id}_B$, then g is injective and f is surjective. In particular, f (and also g) is bijective, if $f \circ g = \mathrm{id}_B$ and $g \circ f = \mathrm{id}_A$. In this case, we call g to be the *inverse* of f and denote this as $g = f^{-1}$. Show by examples that both the conditions $f \circ g = \mathrm{id}_B$ and $g \circ f = \mathrm{id}_A$ are necessary for f to be bijective.

2.3 Let $f : A \to B$ a map from a finite set A to a finite set B. Prove that
(a) $\#A \leqslant \#B$, if f is injective,
(b) $\#A \geqslant \#B$, if f is surjective, and
(c) $\#A = \#B$, if f is bijective.

2.4 Let A be a finite set and let $f : A \to A$ be a map. Show that the following conditions are equivalent.
(1) f is injective.
(2) f is surjective.
(3) f is bijective.
Show by examples that this equivalence need not hold, if A is an infinite set.

2.5 Let A and B be two arbitrary sets, $f : A \to B$ a map, $A' \subseteq A$ and $B' \subseteq B$. We define $f(A') := \{y \in B \mid y = f(x) \text{ for some } x \in A'\}$ and $f^{-1}(B') := \{x \in A \mid f(x) \in B'\}$. Show that:

(a) If $A' \subseteq A'' \subseteq A$, then $f(A') \subseteq f(A'')$.
(b) If $B' \subseteq B'' \subseteq B$, then $f^{-1}(B') \subseteq f^{-1}(B'')$.
(c) $f^{-1}(f(A')) \supseteq A'$.
(d) $f(f^{-1}(B')) \subseteq B'$.
(e) $f(f^{-1}(f(A'))) = f(A')$.
(f) $f^{-1}(f(f^{-1}(B'))) = f^{-1}(B')$.

2.6 Russell's paradox A collection $\mathcal{C}$ is called *ordinary*, if $\mathcal{C}$ is not a member of $\mathcal{C}$. A collection which is not ordinary is called *extraordinary*. Show that the collection of all ordinary collections is neither ordinary nor extraordinary.

2.7 Let A_i, $i \in I$, be a family of sets (not necessarily pairwise disjoint). For each $i \in I$, consider the set $B_i := \{(a, i) \mid a \in A_i\} = A_i \times \{i\}$. Show that the family B_i, $i \in I$, are pairwise disjoint. The union $\bigcup_{i \in I} B_i$ is called the *disjoint union* of A_i, $i \in I$.

2.3 Groups

So far we have studied sets as unordered collections. However things start getting interesting if we define one or more binary operations on sets. Such operations define *structures* on sets and we compare different sets in light of their respective structures. Groups are the first (and simplest) examples of sets with binary operations.

Definition 2.1 A *binary operation* on a set A is a map from $A \times A$ to A. If $\diamond$ is a binary operation on A, it is customary to write $a \diamond a'$ to denote the image of (a, a') (under $\diamond$). □

For example, addition, subtraction and multiplication are all binary operations on $\mathbb{Z}$ (or $\mathbb{R}$ or $\mathbb{C}$). Subtraction is not a binary operation on $\mathbb{N}$, since, for example, $2 - 3$ is not an element of $\mathbb{N}$. Division is not a binary operation on $\mathbb{R}$, since division by zero is not defined. Division is a binary operation on $\mathbb{R}^*$.

2.3.1 Definition and Basic Properties

Definition 2.2 A *group*[1] $(G, \diamond)$ is a set G together with a binary operation $\diamond$ on G, that satisfy the following three conditions:

(1) **Associativity** $(a \diamond b) \diamond c = a \diamond (b \diamond c)$ for all $a, b, c \in G$.

(2) **Identity element** There exists a (unique) element $e \in G$ such that $e \diamond a = a \diamond e = a$ for all $a \in G$. The element e is called the *identity* of G.

[1] *In binary operations and algebras generally there is a morass of terminology which reflects on the literacy of the promulgators. Starting for example with a poor choice, namely "group", we now have "semigroup" (why?), "loop"(why?), "groupoid", and "partial groupoid". ... Among other poor choices are "ring", "field", "ideal", "category theory", and "universal algebra". "Ideal" was used by Dedekind in a sense which made sense to mathematicians of that day but it does not today. "Field" can best be labeled as ridiculous. As to categories of category theory, the concept of category is too broad for that reduction. It is not good taste to take such a term and place it in restricted surroundings.*

— Preston C. Hammer

(3) **Inverse** For each $a \in G$, there exists a (unique) element $b \in G$ such that $a \diamond b = b \diamond a = e$. The element b is called the *inverse* of a.

If, in addition, we assume that

(4) **Commutativity** $a \diamond b = b \diamond a$ for all $a, b \in G$,

then G is called a *commutative* or an *Abelian* group. □

A group $(G, \diamond)$ is also written in short as G, when the operation $\diamond$ is understood from the context. More often than not, the operation $\diamond$ is either addition $(+)$ or multiplication $(\cdot)$ in which cases we also say that G is respectively an additive or a multiplicative group. For a multiplicative group, we often omit the multiplication sign and denote $a \cdot b$ simply as ab. The identity in an additive group is usually denoted by 0, whereas that in a multiplicative group by 1. The inverse of an element a in these cases are denoted respectively by $-a$ and a^{-1}. Groups written additively are usually Abelian, but groups written multiplicatively need not be so.

Note that associativity allows us to write $a \diamond b \diamond c$ unambiguously to represent $(a \diamond b) \diamond c = a \diamond (b \diamond c)$. More generally, if $a_1, \ldots, a_n \in G$, then $a_1 \diamond \cdots \diamond a_n$ represents a unique element of the group irrespective of how we insert brackets to compute the element $a_1 \diamond \cdots \diamond a_n$.

Example 2.1

(1) The set $\mathbb{Z}$ is an Abelian group under addition. The identity is 0 and the inverse of a is $-a$. Note, however, that $\mathbb{Z}$ is not a group under multiplication, because though it contains the multiplicative identity 1, multiplicative inverse is not defined for all elements in $\mathbb{Z}$ except ± 1.

(2) The set $\mathbb{Q}^*$ of non-zero rational numbers is a group under multiplication. The identity is $1 = 1/1$ and the inverse of a/b is b/a.

(3) For a set A, the set of all bijective functions $A \to A$ is a group under composition of functions. The identity element is id_A and the inverse of f is denoted by f^{-1}. (See also Exercise 2.2.) This group is not Abelian in general.

(4) The set $M_{m,n}(\mathbb{R})$ of all $m \times n$ matrices with entries from $\mathbb{R}$ is a group under matrix addition. On the other hand, the set $GL_n(\mathbb{R})$ of all $n \times n$ invertible matrices over $\mathbb{R}$ is a group under matrix multiplication and is called the *general linear group*. Note that $GL_n(\mathbb{R})$ is another example of a group that is not Abelian (for $n > 1$).

(5) A group G is called *finite*, if G as a set consists of (only) finitely many elements. Finite groups play an extremely important role in cryptography. Here is our first example of finite groups: Let n be an integer $\geqslant 2$. The set

$$\mathbb{Z}_n := \{0, \ldots, n-1\}$$

is a group under addition modulo n (that is, add (and subtract) two elements in $\mathbb{Z}_n$ as integers and if the result is not in $\mathbb{Z}_n$, take the remainder of division by n). For this group, the identity element is 0 and $-a = n - a$ for $a \neq 0$ and $-0 = 0$. (See Example 2.3 for a formal definition of $\mathbb{Z}_n$.)

(6) For an integer $n \geqslant 2$, define the set

$$\mathbb{Z}_n^* := \{a \mid 0 \leqslant a < n, a \text{ is relatively prime to } n\}.$$

If n is prime, then $\mathbb{Z}_n^* = \{1, \ldots, n-1\}$. The set $\mathbb{Z}_n^*$ is a group under multiplication modulo n with identity 1. We need little more machinery than introduced so far in order to prove that every element $a \in \mathbb{Z}_n^*$ has a multiplicative inverse modulo n. Other group axioms are easy to check. □

Proposition 2.1 Let $(G, \diamond)$ be a group and let $a, b, c \in G$. Then $a \diamond b = a \diamond c$ implies $b = c$. Similarly, $a \diamond c = b \diamond c$ implies $a = b$. These statements are commonly known as (left and right) *cancellation laws*.

Proof We prove only the left cancellation law. The proof of the other law is similar. Let e denote the identity of G and d the inverse of a. Then $b = e \diamond b = (d \diamond a) \diamond b = d \diamond (a \diamond b) = d \diamond (a \diamond c) = (d \diamond a) \diamond c = e \diamond c = c$. ■

2.3.2 Subgroups, Cosets and Quotient Groups

Definition 2.3 Let $(G, \diamond)$ be a group. Then a subset H of G is called a *subgroup* of G, if H is a group under the operation $\diamond$ inherited from G. For a subset H of G to be a subgroup, it is necessary and sufficient that H is closed under the operation $\diamond$ and under inverse. Any subgroup of an Abelian group is also Abelian. □

Example 2.2

(1) For any group G with identity element e, the subsets $\{e\}$ and G are subgroups of G. They are called the *trivial subgroups* of G.

(2) For an integer $n \geqslant 2$, the set of all integral multiples of n is an additive subgroup of $\mathbb{Z}$ and is denoted by $n\mathbb{Z}$.

(3) The set $SL_n(\mathbb{R})$ consisting of all $n \times n$ real matrices of determinant 1 is a subgroup of $GL_n(\mathbb{R})$ and is commonly referred to as the *special linear group*.

(4) Note that though $\mathbb{Z}_n$ in Example 2.1 is a subset of $\mathbb{Z}$, it is not a subgroup of $\mathbb{Z}$, since it is not closed under the addition of $\mathbb{Z}$. It is a group under addition modulo n which is not the same as integer addition. □

Let $(G, \diamond)$ be a group. For subsets A and B of G, we denote by $A \diamond B$ the set $\{a \diamond b \mid a \in A \text{ and } b \in B\}$. In particular, if $A = \{a\}$ (resp. $B = \{b\}$), then $A \diamond B$ is denoted by $a \diamond B$ (resp. $A \diamond b$). Note that the sets $A \diamond B$ and $B \diamond A$ are not necessarily equal. If G is Abelian, then $A \diamond B = B \diamond A$.

Definition 2.4 Let $(G, \diamond)$ be a group, H a subgroup of G and $a \in G$. The set $a \diamond H$ is called the *left coset* of a with respect to H and the set $H \diamond a$ is called the *right coset* of a with respect to H. If G is Abelian, then a left coset is naturally a right coset and vice versa. In that case, we call $a \diamond H$ (or $H \diamond a$) simply a *coset*. □

From now onward, we consider left cosets only and call them cosets. If the underlying group is Abelian, then they are the same thing. The theory of right cosets can be

parallelly developed, but we choose to omit that here. For simplicity, we also assume that the group G is a multiplicative group, so that the operation $\diamond$ would be replaced by $\cdot$ (or by mere juxtaposition).

Proposition 2.2 Let G be a (multiplicative) group and H a subgroup of G. Then, the cosets aH, $a \in G$, partition G. Two cosets aH and bH are equal if and only if $a^{-1}b \in H$. There is a bijective map from aH to bH for every $a, b \in G$.

Proof We define a relation $\sim$ on G such that $a \sim b$ if and only if $a^{-1}b \in H$. Clearly, $a \sim a$. Now $a \sim b$ implies $a^{-1}b \in H$, so that $b^{-1}a = (a^{-1}b)^{-1} \in H$ (See Exercise 2.8), that is, $b \sim a$. Finally, $a \sim b$ and $b \sim c$ imply $a \sim c$, since $a^{-1}c = (a^{-1}b)(b^{-1}c)$. Thus $\sim$ is an equivalence relation on G and hence by Theorem 2.1 produces a partition of G. We now show that the equivalence class $[a]$ of $a \in G$ is the coset aH. This follows from that $b \in [a] \iff a^{-1}b \in H \iff a^{-1}b = h$ for some $h \in H \iff b = ah$ for some $h \in H \iff b \in aH$.

Now we define a map $\varphi : aH \to bH$ by $ah \mapsto bh$ for every $h \in H$. The map is clearly surjective. Injectivity of φ follows from the left cancellation law (Proposition 2.1). Hence φ is bijective. ■

The following theorem is an important corollary to the last proposition.

Theorem 2.2 **Lagrange's theorem** Let G be a finite group and H a subgroup of G. Then, the cardinality of G is an integral multiple of the cardinality of H.

Proof From Proposition 2.2, the cosets form a partition of G and there is a bijective map from one coset to another. Hence by Exercise 2.3 all cosets have the same cardinality. Finally, note that H is the coset of the identity element. ■

Definition 2.5 Let G be a group and H a subgroup of G. The number of distinct cosets of H in G is called the *index* of H in G and is denoted by $[G : H]$. If G is finite, then $[G : H] = \#G/\#H$. □

Definition 2.6 Let H be a subgroup of a (multiplicative) group G. Then H is called a *normal subgroup* of G, if $(aH)(bH) = (abH)$ for all $a, b \in G$. It is clear that any subgroup H of an Abelian group G satisfies this condition and hence is normal.

If H is a normal subgroup of a group G, then the cosets aH, $a \in G$, form a group with multiplication defined by $(aH)(bH) = (abH)$. This group is called the *quotient group* of G with respect to H and is denoted by G/H. □

Example 2.3

(1) Let n be an integer $\geqslant 2$. The subgroup $n\mathbb{Z}$ of $(\mathbb{Z}, +)$ (Example 2.2) is normal, since $\mathbb{Z}$ is Abelian. The coset of $i \in \mathbb{Z}$ is the set $i + n\mathbb{Z} = \{i + jn \mid j \in \mathbb{Z}\}$. The quotient group $\mathbb{Z}/n\mathbb{Z}$ is denoted as $\mathbb{Z}_n$ and is essentially the same as the group $\{0, 1, \ldots, n-1\}$ with the operation of addition modulo n (Example 2.1).

(2) For any group G with identity e, the trivial subgroups G and $\{e\}$ are normal. G/G is a group with a single element, whereas $G/\{e\}$ is essentially the same as the group G. □

2.3.3 Homomorphisms

Definition 2.7 Let $(G, \diamond)$ and $(G', \odot)$ be groups. A function $f : G \to G'$ is called a *homomorphism* (of groups), if $f(a \diamond b) = f(a) \odot f(b)$ for all $a, b \in G$, that is, if f commutes with the group operations of G and G'.

A group homomorphism $f : G \to G'$ is called an *isomorphism*, if there exists a group homomorphism $g : G' \to G$ such that $g \circ f = \mathrm{id}_G$ and $f \circ g = \mathrm{id}_{G'}$. It can be easily seen that a homomorphism $f : G \to G'$ is an isomorphism if and only if f is bijective as a function.[2] If there exists an isomorphism $f : G \to G'$, we say that the groups G and G' are *isomorphic* and write $G \cong G'$.

A homomorphism f from G to itself is called an *endomorphism* (of G). An endomorphism which is also an isomorphism is called an *automorphism*. The set of all automorphisms of a group G is a group under function composition. We denote this group by $\mathrm{Aut}\, G$. □

Example 2.4

(1) The *canonical inclusion* $a \mapsto a/1$ is a group homomorphism from $(\mathbb{Z}, +)$ to $(\mathbb{Q}, +)$. More generally, if H is a subgroup of G, then the map $h \mapsto h$ for all $h \in H$ is a group homomorphism. In particular, the identity map on any group G is an automorphism of G (and is the identity element of the group $\mathrm{Aut}\, G$).

(2) For a (multiplicative) group G and a normal subgroup H, the map $G \to G/H$ that takes $a \in G$ to its coset aH is a surjective group homomorphism. It is called the *canonical surjection* of G onto G/H. For example, the map that takes a to its remainder of division by n ($\geqslant 2$) is a canonical surjection from the additive group $\mathbb{Z}$ to the quotient group $\mathbb{Z}_n = \mathbb{Z}/n\mathbb{Z}$. (Also see Examples 2.1, 2.2 and 2.3.)

(3) The map that takes a complex number $z = a + \mathrm{i}b$ to its conjugate $\bar{z} = a - \mathrm{i}b$ is a group automorphism of both $(\mathbb{C}, +)$ and $(\mathbb{C}^*, \cdot)$. □

Proposition 2.3 Let f be a group homomorphism from $(G, \diamond)$ to $(G', \odot)$. Let e and e' denote the identity elements of G and G' respectively. Then $f(e) = e'$. If $a, b \in G$ and $c, d \in G'$ satisfy $a \diamond b = e$, $c \odot d = e'$ and $f(a) = c$, then $f(b) = d$.

Proof We have $e' \odot f(e) = f(e) = f(e \diamond e) = f(e) \odot f(e)$, so that by right cancellation $f(e) = e'$. To prove the second assertion we note that $c \odot d = e' = f(e) = f(a \diamond b) = f(a) \odot f(b) = c \odot f(b)$. Thus $f(b) = d$. ■

Definition 2.8 With the notations of the last proposition we define the *kernel* of f to be the following subset of G:

$$\mathrm{Ker}\, f := \{a \in G \mid f(a) = e'\}.$$

We also define the *image* of f to be the subset

$$\mathrm{Im}\, f := \{b \in G' \mid \exists a \in G \text{ with } b = f(a)\}$$

of G'. Then we have the following important theorem. □

[2] If $f : G \to G'$ is a bijective homomorphism, its inverse $f^{-1} : G' \to G$ is bijective as a function. However, it is not obvious that f^{-1} has to be a group homomorphism. We are lucky here; f^{-1} is.

Theorem 2.3 **Isomorphism theorem** $\operatorname{Ker} f$ is a normal subgroup of G, $\operatorname{Im} f$ is a subgroup of G', and $G/\operatorname{Ker} f \cong \operatorname{Im} f$.

Proof In order to simplify notations, let us assume that G and G' are multiplicatively written groups. For $u, v \in \operatorname{Ker} f$, we have $f(uv^{-1}) = f(u)(f(v))^{-1} = e'$, that is, $uv^{-1} \in \operatorname{Ker} f$. By Exercise 2.8, $\operatorname{Ker} f$ is a subgroup of H. We now show that it is normal. Note that for $a \in G$ and $u \in \operatorname{Ker} f$ we have $f(aua^{-1}) = f(a)f(u)f(a^{-1}) = e'$, that is, $aua^{-1} \in \operatorname{Ker} f$, since $f(u) = e'$ and $f(a^{-1}) = f(a)^{-1}$. By Exercise 2.10, $\operatorname{Ker} f$ is a normal subgroup of G. Now let $a' = f(a)$ and $b' = f(b)$ be arbitrary elements of $\operatorname{Im} f$. Then, $f(ab^{-1}) = a'(b')^{-1}$, that is, $a'(b')^{-1} \in \operatorname{Im} f$. Thus, by Exercise 2.8 $\operatorname{Im} f$ is a subgroup of G'.

Now define a map $\varphi : G/\operatorname{Ker} f \to \operatorname{Im} f$ that takes $a \operatorname{Ker} f \mapsto f(a)$. Let $a \operatorname{Ker} f = b \operatorname{Ker} f$. Then by Proposition 2.2, $a^{-1}b \in \operatorname{Ker} f$, that is, $b = au$ for some $u \in \operatorname{Ker} f$. But then $f(b) = f(au) = f(a)f(u) = f(a)e' = f(a)$. This shows that the map φ is well-defined. It is easy to check that φ is a group homomorphism. Now $\varphi(a \operatorname{Ker} f) = \varphi(b \operatorname{Ker} f)$ implies $f(a) = f(b)$, that is, $f(a^{-1}b) = e'$, that is, $a^{-1}b \in \operatorname{Ker} f$, that is, $a \operatorname{Ker} f = b \operatorname{Ker} f$. Thus φ is injective. It is clearly surjective. Thus φ is bijective and hence an isomorphism from $G/\operatorname{Ker} f$ to $\operatorname{Im} f$. ■

2.3.4 Generators and Orders

Definition 2.9 Let G be a group. In this section, we assume, unless otherwise stated, that G is multiplicatively written and has identity e. Let a_i, $i \in I$, be a family of elements of G. Consider the subset H of G defined as

$$H := \{b_{i_1} \cdots b_{i_r} \mid r \in \mathbb{N}_0 \text{ and each } b_{i_j} = a_{i_j} \text{ or } a_{i_j}^{-1} \text{ for some } i_j \in I\}$$

with the empty product (corresponding to $r = 0$) being treated as e. It is easy to check that H is a subgroup of G and contains all a_i, $i \in I$. We call H to be the *subgroup generated by* a_i, $i \in I$, or that the elements a_i, $i \in I$, generate H. H is called *finitely generated*, if it is generated by finitely many elements. In particular, H is called *cyclic*, if it is generated by a single element. If H is cyclic and generated by $g \in H$, then g is called a *generator* or a *primitive element* of H. Note that, in general, a cyclic subgroup has more than one generators (Exercise 2.47). □

Example 2.5

(1) The additive groups $\mathbb{Z}$ and $\mathbb{Z}_n$ are generated by 1 and hence are cyclic. The multiplicative group $\mathbb{Z}_n^*$ ($n \geqslant 2$) is cyclic if and only if n is 2, 4, p^r or $2p^r$, where p is an odd prime and $r \in \mathbb{N}$ (See Exercise 2.50). A generator of $\mathbb{Z}_n^*$ for such an n is often called a *primitive root* modulo n.

(2) The group $(\mathbb{Q}^*, \cdot)$ is generated by the "primes" $p/1$, $p \in \mathbb{P}$, and -1.

(3) Let G be a multiplicative group (not necessarily Abelian) with identity e and let $a \in G$. Then the subgroup H generated by a is the set of elements of the form a^r, $r \in \mathbb{Z}$, and is always Abelian. If H is finite, then the elements a^r, $r \in \mathbb{Z}$, cannot be all distinct, that is, $a^s = a^t$ for some $s, t \in \mathbb{Z}$, $s > t$. Then $a^{s-t} = e$, where $s - t > 0$. Now $a^{-1} = a^{s-t-1}$ and, more generally, $a^{-k} = a^{k(s-t-1)}$.

Thus we may consider H to consist of non-negative powers of a only. Let $n = \min\{r \in \mathbb{N} \mid a^r = e\}$. It is easy to see that $H = \{a^r \mid r = 0, \ldots, n-1\}$. □

Definition 2.10 Let G be a *finite* group with identity e. The *order* of G is defined to be the cardinality of the set G and is denoted by $\operatorname{ord} G$. The *order* of an element $a \in G$ is the cardinality of the subgroup of G generated by a and is denoted by $\operatorname{ord}_G a$ or simply by $\operatorname{ord} a$, when G is understood from the context. □

With these notations we prove the following important proposition.

Proposition 2.4 The order $m := \operatorname{ord}_G a$ of $a \in G$ is the smallest of the positive integers r for which $a^r = e$. If $n = \operatorname{ord} G$, then n is an integral multiple of m. In particular, $a^n = e$.

Proof Let H be the (cyclic) subgroup of G generated by a. Then by Example 2.5 $H = \{a^r \mid r = 0, \ldots, m-1\}$ and m is the smallest of the positive integers r for which $a^r = e$. By Lagrange's theorem (Theorem 2.2), n is an integral multiple of m. That is, $n = km$ for some $k \in \mathbb{N}$. But then $a^n = (a^m)^k = e^k = e$. ■

Lemma 2.1 Let G be a finite cyclic group. Then any subgroup of G is also cyclic.

Proof Let G be generated by g and $\operatorname{ord} G = n$. Then $G = \{g^r \mid r = 0, \ldots, n-1\}$. The subgroup $\{e\}$ of G is clearly cyclic. For an arbitrary subgroup $H \neq \{e\}$ of G, define $k := \min\{r > 0 \mid g^r \in H\}$. Now take any $g^r \in H$ and write $r = qk + \delta$, where q and δ are respectively the quotient and remainder of division of r by k with $0 \leqslant \delta < k$. Then $g^r = (g^k)^q g^\delta$ and so $g^\delta \in H$. The minimality of k implies that $\delta = 0$, that is, $g^r = (g^k)^q$. ■

Proposition 2.5 Let G be a finite cyclic multiplicative group with identity e and let H be a subgroup of order m. Then an element $a \in G$ is an element of H if and only if $a^m = e$.

Proof If $a \in H$, then $a^m = e$ by Proposition 2.4. Conversely, assume that $a^m = e$, but $a \notin H$. Let K be the subgroup of G generated by the elements of H and by a. By Lemma 2.1, K is cyclic. By assumption, K contains more than m elements (since $H \cup \{a\} \subseteq K$). But every element of K has order dividing m, a contradiction. ■

Finite cyclic groups play a crucial role in public-key cryptography. To see how, let G be a group which is finite, cyclic with generator g and multiplicatively written. Given $r \in \mathbb{N}$ one can compute g^r using $\leqslant 2 \lg r + 2$ group multiplications (See Algorithms 3.9 and 3.10). This means that if it is *easy* to multiply elements of G, then it is also *easy* to compute g^r. On the other hand, there are certain groups for which it is very *difficult* to find out the integer r from the knowledge of g and g^r, even when one is certain that such an integer exists. This is the basic source of security in many cryptographic protocols, like those based on finite fields, elliptic and hyperelliptic curves.

* 2.3.5 Sylow's Theorem

Sylow's theorem is a powerful tool for studying the structure of finite groups. Recall that if G is a finite group of order n and if H is a subgroup of G of order m, then by Lagrange's theorem m divides n. But given any divisor m' of n, there need not exist a subgroup of G of order m'. However, for certain special values of m', we can prove

the existence of subgroups of order m'. Sylow's theorem considers the case that m' is a power of a prime.

Definition 2.11 Let G be a finite group of cardinality n and let p be a prime. If $n = p^r$ for some $r \in \mathbb{N}$, we call G a *p-group*. More generally, let p be a prime divisor of n. Then a *p-subgroup* of G is a subgroup H of G such that H is a p-group. If H is a p-subgroup of G with cardinality p^r for some $r \in \mathbb{N}$, then p^r divides n. Moreover, if p^{r+1} does not divide n, then H is called a *p-Sylow subgroup* of G. □

We shortly prove that p-Sylow subgroups always exist. Before doing that, we prove a simpler result.

Theorem 2.4 **Cauchy's theorem** Let G be a finite group and p a prime dividing $\operatorname{ord} G$. Then G has a subgroup of order p.

Proof Let $n := \operatorname{ord} G$. Note that if we can find an element $a \in G$ such that $\operatorname{ord} a = p$, then the subgroup generated by a is the desired subgroup. To do that consider the set $\mathfrak{S}$ consisting of all p-tuples $(a_1, \ldots, a_p)$ with $a_i \in G$ such that $a_1 \cdots a_p = e$. $\mathfrak{S}$ consists of n^{p-1} elements, since we can choose $a_1, \ldots, a_{p-1}$ arbitrarily and independently from G and for each such choice of $a_1, \ldots, a_{p-1}$ the value of $a_p = (a_1 \cdots a_{p-1})^{-1}$ gets fixed. Since p divides n, it follows that p divides $\#\mathfrak{S}$ too. Now we define a relation $\sim$ on $\mathfrak{S}$ by $(a_1, \ldots, a_p) \sim (b_1, \ldots, b_p)$ if and only if $(b_1, \ldots, b_p) = (a_i, \ldots, a_p, a_1, \ldots, a_{i-1})$ for some $i \in \{1, \ldots, p\}$ (that is, $(b_1, \ldots, b_p)$ is a cyclic shift of $(a_1, \ldots, a_p)$). It is easy to see that $\sim$ is an equivalence relation on $\mathfrak{S}$. The equivalence class of $(a_1, \ldots, a_p)$ contains 1 or p elements depending on whether $a_1 = \cdots = a_p$ or not. Let r and s be the the number of equivalence classes containing 1 and p elements of $\mathfrak{S}$ respectively. Then $\#\mathfrak{S} = r + sp$, so that p divides r. Since the equivalence class of $(e, \ldots, e)$ contains only one element, we must have $r \geqslant 1$, that is, $r \geqslant p$. This, in turn, proves the existence of $a \in G$, $a \neq e$, such that $(a, \ldots, a) \in \mathfrak{S}$. But then $a^p = e$. ■

Now we are in a position to prove the general theorem.

Theorem 2.5 **Sylow's theorem** Let G be a finite group of order n and let p be a prime dividing n. Then there exists a p-Sylow subgroup of G.

Proof We proceed by induction on n. If $n = p$, then G itself is a p-Sylow subgroup of G. So we assume $n > p$ and write $n = p^r m$, where p does not divide m. If $r = 1$, then the theorem follows from Cauchy's theorem (Theorem 2.4). So we assume $r > 1$ and consider the class equation of G, namely, $\#G = \#Z(G) + \sum[G : C(a)]$ (See Exercise 2.16). If p does not divide $[G : C(a)]$ for some $a \notin Z(G)$, then $\#C(a) = \#G/[G : C(a)] = p^r m' < \#G$ for some $m' < m$. By induction, $C(a)$ has a p-Sylow subgroup which is also a p-Sylow subgroup of G. On the other hand, if p divides $[G : C(a)]$ for all $a \notin Z(G)$, then p divides $\#Z(G)$, as can be easily seen from the class equation. We apply Cauchy's theorem on $Z(G)$ to obtain a subgroup H of $Z(G)$ with $\#H = p$. By Exercise 2.16(b), H is a normal subgroup of G and we consider the canonical surjection $\mu : G \to G/H$. Since $\#(G/H) = p^{r-1}m < n$ and $r > 1$, by induction G/H has a p-Sylow subgroup, say K. But then $\mu^{-1}(K)$ is a p-Sylow subgroup of G. ■

Note that if H is a p-Sylow subgroup of G and $g \in G$, then gHg^{-1} is also a p-Sylow subgroup of G. The converse is also true, that is, if H and H' are two p-Sylow

subgroups of G, then there exists a $g \in G$ such that $H' = gHg^{-1}$. We do not prove this assertion here, but mention the following important consequence of it. If G is *Abelian*, then $H' = gHg^{-1} = gg^{-1}H = H$, that is, there is only one p-Sylow subgroup of G. If G is Abelian and $\#G = p_1^{\alpha_1} \cdots p_t^{\alpha_t}$ with pairwise distinct primes p_i and with $\alpha_i \in \mathbb{N}$, then G is the internal direct product of its p_i-Sylow subgroups, $i = 1, \ldots, t$ (Exercises 2.17 and 2.19).

EXERCISE SET 2.3

2.8 Let G be a multiplicatively written group (not necessarily Abelian). Prove the following assertions.
(a) For all elements $a, b \in G$, we have $(ab)^{-1} = b^{-1}a^{-1}$ and $(a^{-1})^{-1} = a$.
(b) A subset H of G is a subgroup of G if and only if $ab^{-1} \in H$ for all $a, b \in H$.

2.9 Let G be a multiplicatively written group and let H and K be subgroups of G. Show that:
(a) $H \cap K$ is a subgroup of G.
(b) $H \cup K$ is a subgroup of G if and only if $H \subseteq K$ or $K \subseteq H$.
(c) HK is a subgroup of G if and only if $HK = KH$. In particular, if K is normal in G, then HK is a subgroup of G.
(d) $G \times G$ is a group and $H \times K$ is a subgroup of $G \times G$.
(e) If $g \in G$, then gHg^{-1} is a subgroup of G.

2.10 **(a)** Let G be a multiplicatively written group and H a subgroup of G. Show that the following conditions are equivalent:
(1) H is a normal subgroup of G.
(2) $ghg^{-1} \in H$ for all $g \in G$ and $h \in H$.
(3) $gHg^{-1} = H$ for all $g \in G$.
(4) $gH = Hg$ for all $g \in G$.
(b) Show that if $[G : H] = 2$, then H is normal.

2.11 Let G be a (multiplicative) group.
(a) Second isomorphism theorem Let H and K be subgroups of G and let K be normal in G. Show that $H/(H \cap K) \cong (HK)/K$. **[H]**
(b) Third isomorphism theorem Let H and K be normal subgroups of G with $H \subseteq K$. Show that $G/K \cong (G/H)/(K/H)$ (where $K/H = \{aH \mid a \in K\}$). **[H]**

2.12 **(a)** Show that the only automorphisms of the group $(\mathbb{Z}, +)$ are the identity map and the map that sends $a \mapsto -a$.
(b) Show that the group of automorphisms of $(\mathbb{Z}_n, +)$ is isomorphic to $(\mathbb{Z}_n^*, \cdot)$.

2.13 Let H be a subgroup of G generated by a_i, $i \in I$. Show that H is the smallest subgroup of G, that contains all of a_i, $i \in I$.

2.14 Let $\varphi : G \to G'$ be a homomorphism of (multiplicative) groups. Show that:
(a) If H is a subgroup of G, then $H' := \varphi(H)$ is a subgroup of G'. If φ is surjective and H is normal, then H' is also normal.

(b) If H' is a subgroup of G', then $H := \varphi^{-1}(H')$ is a subgroup of G. If H' is normal, then H is also normal. If φ is surjective and H is normal, then H' is also normal.
(c) Correspondence theorem Let H be a normal subgroup of G. Then the subgroups (resp. normal subgroups) of G/H are in one-to-one correspondence with the subgroups (resp. normal subgroups) of G, that contain H. **[H]**

2.15 Let G be a cyclic group. Show that G is isomorphic to $\mathbb{Z}$ or to $\mathbb{Z}_n$ for some $n \in \mathbb{N}$ depending on whether G is infinite or finite.

2.16 Let G be a finite (multiplicative) group (not necessarily Abelian).
(a) We define the *centre* of G to be the set $Z(G) := \{a \in G \mid ab = ba \text{ for all } b \in G\}$. Show that $Z(G)$ is a subgroup of G.
(b) If $H \subseteq Z(G)$ is a subgroup of G, show that H is a normal subgroup of G.
(c) The *centralizer* of $a \in G$ is defined to be the set $C(a) := \{b \in G \mid ab = ba\}$. Show that $C(a)$ is a subgroup of G. Show also that $C(a) = G$ if and only if $a \in Z(G)$.
(d) Define a relation $\sim$ on G by $a \sim b$ if and only if $b = gag^{-1}$ for some $g \in G$. Show that $\sim$ is an equivalence relation on G. We say that the elements a and b of G are *conjugate*, if the equivalence classes $[a]$ and $[b]$ are the same. The equivalence classes are called the *conjugacy classes* of G.
(e) Show that the cardinality of the conjugacy class of $a \in G$ is equal to the index $[G : C(a)]$.
(f) Deduce the *class equation* of G, that is, $\#G = \#Z(G) + \sum[G : C(a)]$, where the sum is over a set of all pairwise non-conjugate $a \notin Z(G)$.

2.17 Let G be a (multiplicative) Abelian group with identity e and order $n = p_1^{\alpha_1} \cdots p_r^{\alpha_r}$, where p_i are distinct primes and $\alpha_i \in \mathbb{N}$. For each i, let H_i be the p_i-Sylow subgroup of G. Show that:
(a) $G = H_1 \cdots H_r$. **[H]**
(b) Every element $g \in G$ can be written *uniquely* as $g = h_1 \cdots h_r$ with $h_i \in H_i$. Moreover, in that case we have $\operatorname{ord}_G g = (\operatorname{ord}_{H_1} h_1) \cdots (\operatorname{ord}_{H_r} h_r)$.
(c) G is cyclic if and only if all of $H_1, \ldots, H_r$ are cyclic.

2.18 Let G be a finite (multiplicative) Abelian group with identity e. Assume that for every $n \in \mathbb{N}$ there are at most n elements x of G satisfying $x^n = e$. Show that G is cyclic. **[H]**

2.19 Let G be a (multiplicative) group and let $H_1, \ldots, H_r$ be normal subgroups of G. If $G = H_1 \cdots H_r$ and every element $g \in G$ can be written uniquely as $g = h_1 \cdots h_r$ with $h_i \in H_i$, then G is called the *internal direct product* of $H_1, \ldots, H_r$. (For example, if G is finite and Abelian, then by Exercise 2.17 it is the internal direct product of its Sylow subgroups.) Show that:
(a) If G is finite, it is the internal direct product of normal subgroups $H_1, \ldots, H_r$ if and only if $G = H_1 \cdots H_r$ and $H_i \cap H_j = \{e\}$ for all i, j, $i \neq j$.
(b) If G is the internal direct product of the normal subgroups $H_1, \ldots, H_r$, then G is isomorphic to the (external) direct product $H_1 \times \cdots \times H_r$. **[H]**

2.20 Let H_i, $i = 1, \ldots, r$, be finite Abelian groups of orders m_i and let $H := H_1 \times \cdots \times H_r$ be their direct product. Show that H is cyclic if and only if each H_i is cyclic and $m_1, \ldots, m_r$ are pairwise coprime.

2.4 Rings

So far we have studied algebraic structures with only one operation. Now we study rings which are sets with two (compatible) binary operations. Unlike groups, these two operations are usually denoted by $+$ and $\cdot$. One can, of course, go for general notations for these operations. However, that generalization doesn't seem to pay much, but complicates matters. We stick to the conventions.

2.4.1 Definition and Basic Properties

Definition 2.12 A *ring* $(R, +, \cdot)$ (or R in short) is a set R together with two binary operations $+$ and $\cdot$ on R such that the following conditions are satisfied. As in the case of multiplicative groups we write ab for $a \cdot b$.

(1) **Additive group** The set R is an Abelian group under $+$. The additive identity is denoted by 0.

(2) $\cdot$ **is associative** $(ab)c = a(bc)$ for every $a, b, c \in R$.

(3) $\cdot$ **is commutative** $ab = ba$ for every $a, b \in R$.

(4) **Multiplicative identity** There is an element (denoted by 1) in R such that $a \cdot 1 = 1 \cdot a = a$ for every $a \in R$. The element 1 is called the *identity* of R.

(5) **Distributivity** The operation $\cdot$ is distributive over $+$, that is, $a(b+c) = ab+ac$ and $(a + b)c = ac + bc$ for every $a, b, c \in R$. □

Notice that it is more conventional to define a ring as an algebraic structure $(R, +, \cdot)$ that satisfies conditions (1), (2) and (5) only. A ring (by the conventional definition) is called a *commutative ring* (resp. a *ring with identity*), if it (additionally) satisfies condition (3) (resp. (4)). As per our definition, a ring is always a commutative ring with identity. Rings that are not commutative or that do not contain the identity element are not used in the rest of the book. So let us be happy with our *unconventional* definition of a ring.[3]

We do not rule out the possibility that $0 = 1$ in R. In that case, for any $a \in R$, we have $a = a \cdot 1 = a \cdot 0 = 0$ (See Proposition 2.6), that is to say, the set R consists of the single element 0. In this case, R is called the *zero ring* and is denoted (by an abuse of notation) by 0.

Finally, note that R is, in general, not a group under multiplication. This is because we do not expect a ring R to contain the multiplicative inverse of every element of R. Indeed the multiplicative inverse of the element 0 exists if and only if $R = 0$.

[3] Cool! But what's circular in a *ring*? Historically, such algebraic structures were introduced by Hilbert to designate a Zahlring (a number ring, see Section 2.13). If α is an algebraic integer (Definition 2.95) and we take a Zahlring of the form $\mathbb{Z}[\alpha] = \{f(\alpha) \mid f(X) \in \mathbb{Z}[X]\}$ and consider the powers $\alpha, \alpha^2, \alpha^3, \ldots$, we eventually get an α^d which can be expressed as a linear combination of the previous (that is, smaller) powers of α. This is perhaps the reason that prompted Hilbert to call such structures "rings". Also see Footnote 1.

Example 2.6

(1) The sets $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$ and $\mathbb{C}$ are all rings under usual addition and multiplication. Each of $\mathbb{Q}$, $\mathbb{R}$ and $\mathbb{C}$ contains the multiplicative inverse of *every* non-zero element, whereas the only elements in $\mathbb{Z}$, that have multiplicative inverses, are ± 1.

(2) Let $\mathbb{Z}_n$ denote the set $\{0, 1, \ldots, n-1\}$ for an integer $n \geqslant 2$. Then $\mathbb{Z}_n$ is a ring under addition and multiplication modulo n. The additive identity is 0 and the multiplicative identity is 1. Later we see a more formal definition of this ring. Recall from Example 2.1 how we have defined the groups $\mathbb{Z}_n$ and $\mathbb{Z}_n^*$ under addition and multiplication modulo n. These groups have a connection with the ring $\mathbb{Z}_n$ as we will shortly see.

(3) Let R be a ring and S a set. The set of all functions $S \to R$ is a ring under pointwise addition and multiplication of functions (that is, if f and g are two such functions, then we define $(f+g)(a) := f(a) + g(a)$ and $(fg)(a) := f(a)g(a)$ for every $a \in S$). The additive (resp. multiplicative) identity in this ring is the constant function 0 (resp. 1).

(4) Let R be a ring. The set $R[X]$ of all polynomials in one indeterminate X and with coefficients from R is a ring. The identity elements in $R[X]$ are the constant polynomials 0 and 1. The addition and multiplication operations in $R[X]$ are the standard ones on polynomials. For a non-zero polynomial $f \in R[X]$, the largest non-negative integer d for which the coefficient of X^d is non-zero is called the *degree* of the polynomial f and is denoted by $\deg f$. The coefficient of $X^{\deg f}$ in f is called the *leading coefficient* of f and is denoted by $\mathrm{lc}(f)$. The degree of the zero polynomial is conventionally taken to be $-\infty$. A non-zero polynomial with leading coefficient 1 is called a *monic polynomial*.

More generally, for $n \in \mathbb{N}$ one can define the ring $R[X_1, \ldots, X_n]$ of multivariate polynomials over R. Polynomial rings are of paramount importance in algebra and number theory. We devote Section 2.6 to a study of these rings.

We also define the ring $R(X)$ of *rational functions* over R, which consists of elements of the form f/g with $f, g \in R[X]$, $g \neq 0$. More generally, the set of elements f/g with $f, g \in R[X_1, \ldots, X_n]$, $g \neq 0$, is a ring denoted $R(X_1, \ldots, X_n)$.

(5) Let R_i, $i \in I$, be a family of rings, and $R := \prod_{i \in I} R_i$ the product of the sets R_i, $i \in I$, that is, the set of all ordered tuples indexed by I. For tuples $(a_i)_{i \in I}$ and $(b_i)_{i \in I}$, define the sum $(a_i)_{i \in I} + (b_i)_{i \in I} := (a_i + b_i)_{i \in I}$ and the product $(a_i)_{i \in I}(b_i)_{i \in I} := (a_i b_i)_{i \in I}$. It is easy to see that R is a ring with identity elements $(0)_{i \in I}$ and $(1)_{i \in I}$. It is called the *direct product* of the rings R_i, $i \in I$. If I is of finite cardinality n and if $R_i = A$ for all $i \in I$, then $\prod_{i \in I} R_i$ is denoted in short by A^n. □

Proposition 2.6 Let R be a ring. For all $a, b \in R$, we have:

(1) $a \cdot 0 = 0 \cdot a = 0$

(2) $a(-b) = (-a)b = -ab$

(3) $(-a)(-b) = ab$

Proof (1) $a \cdot 0 = a \cdot (0 + 0) = a \cdot 0 + a \cdot 0$, so that $a \cdot 0 = 0$. Similarly, $0 \cdot a = 0$.

(2) By (1), $0 = a \cdot 0 = a(b + (-b)) = ab + a(-b)$, that is, $a(-b) = -ab$. Similarly, $(-a)b = -ab$.

(3) $(-a)(-b) = -(a(-b)) = -(-ab) = ab$. ■

Definition 2.13 Let R be a ring.

(1) An element $a \in R$ is called a *zero-divisor* of R, if $ab = 0$ for some $b \in R$, $b \neq 0$. By this definition, 0 is a zero-divisor of R, unless $R = 0$. The elements 0, 3, 5, 6, 9, 10 and 12 are all the zero-divisors of $\mathbb{Z}_{15}$.

(2) An element $a \in R$ is called a *unit* of R, if there exists an element $b \in R$ such that $ab = 1$. The elements 1 and -1 are units in any ring. It is easy to see that an element cannot be simultaneously a zero-divisor and a unit. The set of all units in a ring R is denoted by R^* and is a group under the multiplication of the ring R (See Exercise 2.21), called the *multiplicative group* or the *group of units* of R. The multiplicative group of the ring $\mathbb{Z}_n$ (Example 2.6) is $\mathbb{Z}_n^*$.

(3) An element $a \in R$ is called *nilpotent*, if $a^k = 0$ for some $k \in \mathbb{N}$. By this definition, 0 is a nilpotent element in any ring. It is also evident that every nilpotent element in a non-zero ring is a zero-divisor. An example of a non-zero nilpotent element in a ring is $4 \in \mathbb{Z}_{16}$.

(4) An element $a \in R$ is called *idempotent*, if $a^2 = a$. In every ring, 0 and 1 are idempotent. The element 6 is idempotent in $\mathbb{Z}_{15}$. It is easy to check that 0 is the only element in a ring, that is both nilpotent and idempotent. □

Definition 2.14 Let R be a ring.

(1) R is called an *integral domain* (or simply a *domain*), if $R \neq 0$ and if R contains no non-zero zero-divisors. Examples of integral domains: $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$, $\mathbb{Z}_{17}$. On the other hand, $3 \cdot 5 = 0$ in $\mathbb{Z}_{15}$, so Z_{15} is not an integral domain.

(2) R is called a *field*, if $R \neq 0$ and if $R^* = R \setminus \{0\}$, that is, if every non-zero element of R is a unit. This means that in a field one can divide any element by any non-zero element. The most common fields are $\mathbb{Q}$, $\mathbb{R}$ and $\mathbb{C}$. Note that $\mathbb{Z}$ is not a field, since, for example, 2 does not have a multiplicative inverse in $\mathbb{Z}$.

(3) A field R with $\#R$ finite is called a *finite field*. The simplest examples of finite fields are the fields $\mathbb{Z}_p$ for prime integers p. In fact, it is easy to see that $\mathbb{Z}_n$ is a field if and only if n is a prime. Finite fields are widely applied for building various cryptographic protocols. See Section 2.9 for a detailed study of finite fields. □

Corollary 2.1 A field is an integral domain.

Proof Recall from Definition 2.13 that an element in a ring cannot be simultaneously a unit and a zero-divisor. ■

Definition 2.15 Let R be a non-zero ring. The *characteristic* of R, denoted $\mathrm{char}\, R$, is the smallest positive integer n such that $1 + 1 + \cdots + 1$ (n times) $= 0$. If no such integer exists, then we take $\mathrm{char}\, R = 0$. □

$\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$ and $\mathbb{C}$ are rings of characteristic zero. If R is a non-zero finite ring, then the elements $1, 1+1, 1+1+1, \ldots$ cannot be all distinct. This shows that there are positive integers m and n, $m < n$, such that $1+1+\cdots+1$ (n times) $= 1+1+\cdots+1$ (m times). But then $1 + 1 + \cdots + 1$ ($n - m$ times) $= 0$. Thus any non-zero finite ring has positive (that is, non-zero) characteristic. If $\mathrm{char}\, R = t$ is finite, then for any $a \in R$ one has $\underbrace{a + \cdots + a}_{(t \text{ times})} = (\underbrace{1 + \cdots + 1}_{(t \text{ times})}) \cdot a = 0$.

In what follows, we will often denote by n the element $1 + 1 + \cdots + 1$ (n times) of any ring. One should not confuse this with the integer n. One can similarly identify a negative integer $-n$ with the ring element $-(1 + 1 + \cdots + 1)(n \text{ times}) = (-1) + (-1) + \cdots + (-1)(n \text{ times})$.

Proposition 2.7 Let R be an integral domain of positive characteristic p. Then p is a prime.

Proof If p is composite, then we can write $p = mn$ with $1 < m < p$ and $1 < n < p$. But then $p = mn = 0$ (in R). Since R is an integral domain, we must have $m = 0$ or $n = 0$ (in R). This contradicts the minimality of p. ■

2.4.2 Subrings, Ideals and Quotient Rings

Just as we studied subgroups of groups, it is now time to study subrings of rings. It, however, turns out that subrings are not that important for the study of rings as the subsets called ideals are. In fact, ideals (and not subrings) help us construct quotient rings. This does not mean that ideals are "normal" subrings! In fact, ideals are, in general, not subrings at all, and conversely. The formal definitions are waiting!

Definition 2.16 Let R be a ring. A subset S of R is called a *subring* of R, if S is a ring under the ring operations of R. In this case, one calls R a *superring* or a *ring extension* of S.

If R and S are both fields, then S is often called a *subfield* of R and R a *field extension* (or simply an *extension*) of S. In that case, one also says that $S \subseteq R$ is a field extension or that R is an extension over S. □

$\mathbb{Z}$ is a subring of $\mathbb{Q}$, $\mathbb{R}$ and $\mathbb{C}$, whereas $\mathbb{Q} \subseteq \mathbb{R}$ and $\mathbb{R} \subseteq \mathbb{C}$ are field extensions.

We demand that a ring always contains the multiplicative identity (Definition 2.12). This implies that if S is a subring of R, then for all integers n, the elements $n \in R$ are also in S (though they need not be pairwise distinct). Similarly, if R and S are fields, then S contains all the elements of the form mn^{-1} for $m, n \in \mathbb{Z}$, $n \neq 0 \in R$ (cf. Exercise 2.26). Thus $2\mathbb{Z}$, the set of all even integers, is not a subring of $\mathbb{Z}$, though it is a subgroup of $(\mathbb{Z}, +)$ (Example 2.2).

Definition 2.17 Let R be a ring. A subset $\mathfrak{a}$ of R is called an *ideal* of R, if $\mathfrak{a}$ is an additive subgroup of $(R, +)$ and if $ra \in \mathfrak{a}$ for all $r \in R$ and $a \in \mathfrak{a}$.[4] □

[4] Kummer introduced the concept of *ideal numbers*. Later Dedekind reformulated Kummer's notion of ideal numbers to define what we now know as *ideals*.

In this book, we will use Gothic letters (usually lower case) like $\mathfrak{a}$, $\mathfrak{b}$, $\mathfrak{c}$, $\mathfrak{p}$, $\mathfrak{q}$ to denote ideals.[5]

The condition for being an ideal is in one sense more stringent than that for being a subring, that is, an ideal has to be closed under multiplication by any element of the entire ring. On the other hand, we do not demand an ideal to necessarily contain the identity element 1. In fact, $2\mathbb{Z}$ is an ideal of $\mathbb{Z}$. Conversely, $\mathbb{Z}$ is a subring of $\mathbb{Q}$ but not an ideal. Subrings and ideals are different things.

Example 2.7

(1) Let R be any ring. The subset $\{0\}$ is an ideal of R, called the *zero ideal* and denoted also by 0. Similarly, the entire ring R is an ideal of R and is called the *unit ideal*. Note that if an ideal $\mathfrak{a}$ contains a unit u of R, then $1 = u^{-1}u$ is also in $\mathfrak{a}$ and so $a = a \cdot 1 \in \mathfrak{a}$ for every $a \in R$. It follows that an ideal $\mathfrak{a}$ of R is the unit ideal if and only if $\mathfrak{a}$ contains a unit—a justification for the name.

(2) The integral multiples of an integer n form an ideal of $\mathbb{Z}$ denoted by $n\mathbb{Z}$. More generally, for any ring R and for any $a \in R$, the set $\{ra \mid r \in R\}$ is an ideal of R and is denoted by Ra or aR or $\langle a \rangle$. Such an ideal is called a *principal ideal*. (See also Definition 2.18.)

(3) Let R be a ring and let $\mathfrak{a}_i$, $i \in I$, be a family of ideals of R. The *intersection* $\bigcap_{i\in I} \mathfrak{a}_i$ is an ideal of R. The set of finite sums the form $\sum_{j=1}^{n} a_{i_j}$ (where $n \in \mathbb{N}_0$ and $a_{i_j} \in \mathfrak{a}_{i_j}$) is an ideal of R. It is called the *sum* of the ideals $\mathfrak{a}_i$, $i \in I$, and is denoted by $\sum_{i\in I} \mathfrak{a}_i$. The union $\bigcup_{i\in I} \mathfrak{a}_i$ is, in general, not an ideal of R. In fact, the sum $\sum_{i\in I} \mathfrak{a}_i$ is the smallest ideal that contains (the set) $\bigcup_{i\in I} \mathfrak{a}_i$. □

Proposition 2.8 The only ideals of a field are the zero ideal and the unit ideal.

Proof By definition, every non-zero element of a field is a unit. ■

Definition 2.18 Let R be a ring and a_i, $i \in I$, a family of elements of R. The *ideal generated by* a_i, $i \in I$, is defined to be the sum $\mathfrak{a} := \sum_{i\in I} Ra_i$ of the principal ideals Ra_i. We denote this as $\mathfrak{a} = \langle a_i, i \in I \rangle$. In this case, we also say that $\mathfrak{a}$ is generated by a_i, $i \in I$. If I is finite, then we say that $\mathfrak{a}$ is *finitely generated*. In particular, if $\#I = 1$, then $\mathfrak{a}$ is a *principal ideal* (See Example 2.7).

An integral domain every ideal of which is principal is called a *principal ideal domain* or *PID* in short. A ring every ideal of which is finitely generated is called *Noetherian*. Thus principal ideal domains are Noetherian. □

Note that an ideal may have different generating sets of varying cardinalities. For example, the unit ideal in any ring is principal, since it is generated by 1. The integers 2 and 3 generate the unit ideal of $\mathbb{Z}$, since $3 - 2 = 1 \in \langle 2, 3 \rangle$. However, neither 2 nor 3 individually generates the unit ideal of $\mathbb{Z}$. Indeed, using Bézout's relation (Proposition 2.16) one can show that for every $n \in \mathbb{N}$ there is a (minimal) generating set of the unit ideal of $\mathbb{Z}$, that contains exactly n integers. Interested readers may try to construct such generating sets as an (easy) exercise.

Theorem 2.6 $\mathbb{Z}$ is a principal ideal domain.

[5] Mathematicians always run out of symbols. Many believe if it is Gothic, it is just ideal!

Proof The zero ideal is generated by 0. Let $\mathfrak{a}$ be a non-zero ideal of $\mathbb{Z}$ and let a be the smallest positive integer contained in $\mathfrak{a}$. We claim that $\mathfrak{a} = \langle a \rangle$. Clearly, $\langle a \rangle \subseteq \mathfrak{a}$. For the converse, take $b \in \mathfrak{a}$. We can write $b = aq + r$, where q and r are the quotient and the remainder of (Euclidean) division of b by a. Now $r = b - aq \in \mathfrak{a}$ and since $0 \leqslant r < a$, by the choice of a we must have $r = 0$, so that $b \in \langle a \rangle$. ■

A very similar argument proves the following theorem. The details are left to the reader. Also see Exercise 2.31.

Theorem 2.7 If K is a field, then $K[X]$ is a principal ideal domain. ■

We now prove a very important theorem:

Theorem 2.8 **Hilbert's basis theorem** If R is a Noetherian ring, then so is the polynomial ring $R[X_1, \ldots, X_n]$ for $n \in \mathbb{N}$. In particular, the polynomial rings $\mathbb{Z}[X_1, \ldots, X_n]$ and $K[X_1, \ldots, X_n]$ are Noetherian, where K is a field.

Proof Using induction on n we can reduce to the case $n = 1$. So we prove that if R is Noetherian, then $R[X]$ is also Noetherian. Let $\mathfrak{a}$ be a non-zero ideal of $R[X]$. Assume that $\mathfrak{a}$ is not finitely generated. Then we can inductively choose non-zero polynomials $f_1, f_2, f_3, \ldots$ from $\mathfrak{a}$ such that for each $i \in \mathbb{N}$ the polynomial f_i is one having the smallest degree in $\mathfrak{a} \setminus \langle f_1, \ldots, f_{i-1} \rangle$. Let $d_i := \deg f_i$. Then $d_1 \leqslant d_2 \leqslant d_3 \leqslant \cdots$. Let a_i denote the leading coefficient of f_i. Consider the ideal $\mathfrak{b} = \langle a_i, i \in \mathbb{N} \rangle$ in R. By hypothesis, $\mathfrak{b}$ is finitely generated, say, $\mathfrak{b} = \langle a_1, \ldots, a_r \rangle$. This, in particular, implies that $a_{r+1} = \sum_{i=1}^{r} \gamma_i a_i$ for some $\gamma_i \in R$. But then the polynomial $g := f_{r+1} - \sum_{i=1}^{r} \gamma_i X^{d_{r+1}-d_i} f_i$ belongs to $\mathfrak{a} \setminus \langle f_1, \ldots, f_r \rangle$, is non-zero and has degree $< d_{r+1}$, a contradiction to the choice of f_{r+1}. Thus $\mathfrak{a}$ must be finitely generated. ■

Two particular types of ideals are very important in algebra.

Definition 2.19 Let R be a ring.

(1) An ideal $\mathfrak{p}$ of R is called a *prime ideal*, if $\mathfrak{p} \neq R$ and if $ab \in \mathfrak{p}$ implies $a \in \mathfrak{p}$ or $b \in \mathfrak{p}$ for $a, b \in R$. The second condition is equivalent to saying that if $a \notin \mathfrak{p}$ and $b \notin \mathfrak{p}$, then the product $ab \notin \mathfrak{p}$. For a prime integer p, the principal ideal $p\mathbb{Z}$ of $\mathbb{Z}$ is prime. On the other hand, for a composite integer n the ideal $n\mathbb{Z}$ of $\mathbb{Z}$ is not prime. For example, $2 \notin 6\mathbb{Z}$ and $3 \notin 6\mathbb{Z}$, but the product $2 \cdot 3 \in 6\mathbb{Z}$.

(2) An ideal $\mathfrak{m}$ of R is called a *maximal ideal*, if $\mathfrak{m} \neq R$ and if for any ideal $\mathfrak{a}$ satisfying $\mathfrak{m} \subseteq \mathfrak{a} \subseteq R$ we have $\mathfrak{a} = \mathfrak{m}$ or $\mathfrak{a} = R$. This means that there are no non-unit ideals of R properly containing $\mathfrak{m}$. All the ideals $p\mathbb{Z}$ of $\mathbb{Z}$ for prime integers p are maximal ideals (Corollary 2.3). Next consider the polynomial ring $R = \mathbb{Z}[X]$ and the principal ideal $\langle X \rangle$ of R. It is easy to see that $\langle X \rangle \subsetneqq \langle X, 2 \rangle \subsetneqq R$. Thus $\langle X \rangle$ is not maximal. □

Prime and maximal ideals can be characterized by some nice equivalent criteria. See Proposition 2.9.

Definition 2.20 Let R be a ring and $\mathfrak{a}$ an ideal of R. Then $\mathfrak{a}$ is a subgroup of the group $(R, +)$. Since $(R, +)$ is Abelian, $\mathfrak{a}$ is a normal subgroup (Definition 2.6). Thus the cosets $a + \mathfrak{a}$, $a \in R$, form an additive Abelian group. We define multiplication on these cosets as $(a + \mathfrak{a})(b + \mathfrak{a}) := ab + \mathfrak{a}$. It is easy to check that this multiplication is well-defined.

Furthermore, the set of these cosets, denoted $R/\mathfrak{a}$, becomes a ring under this addition and multiplication. The ring $R/\mathfrak{a}$ is called the *quotient ring* of R with respect to $\mathfrak{a}$.

We say that two elements $a, b \in R$ are *congruent* modulo an ideal $\mathfrak{a}$ (of R) and write $a \equiv b \pmod{\mathfrak{a}}$, if $a - b \in \mathfrak{a}$. Thus $a \equiv b \pmod{\mathfrak{a}}$ if and only if a and b lie in the same coset of $\mathfrak{a}$, that is, $a + \mathfrak{a} = b + \mathfrak{a}$. □

Example 2.8

(1) For any ring R, the quotient ring $R/0$ is essentially the same as R and the quotient ring R/R is the zero ring.

(2) The ring $\mathbb{Z}_n$ of Example 2.6 is formally defined to be the quotient ring $\mathbb{Z}/n\mathbb{Z}$. Convince yourself that both these definitions are equivalent. □

Proposition 2.9 Let R be a ring and $\mathfrak{a}$ an ideal of R.

(1) $\mathfrak{a}$ is a prime ideal of R if and only if $R/\mathfrak{a}$ is an integral domain.
(2) $\mathfrak{a}$ is a maximal ideal of R if and only if $R/\mathfrak{a}$ is a field.

Proof

(1) Let $a, b \in R$ be arbitrary. Then $\mathfrak{a}$ is prime $\iff$ $ab \in \mathfrak{a}$ implies $a \in \mathfrak{a}$ or $b \in \mathfrak{a}$ $\iff$ $ab + \mathfrak{a} = (a + \mathfrak{a})(b + \mathfrak{a}) = 0$ implies $a + \mathfrak{a} = 0$ or $b + \mathfrak{a} = 0$ $\iff$ $R/\mathfrak{a}$ is an integral domain.

(2) Let $\mathfrak{a}$ be a maximal ideal. Choose $b + \mathfrak{a} \neq 0 + \mathfrak{a}$. Then $b \notin \mathfrak{a}$. Consider the ideal $\mathfrak{b} := \mathfrak{a} + Rb$. Since $\mathfrak{a}$ is maximal, we must have $\mathfrak{b} = R$. This means that $a + cb = 1$ for some $a \in \mathfrak{a}$ and $c \in R$. Then $(c + \mathfrak{a})(b + \mathfrak{a}) = 1 + \mathfrak{a}$ which implies that $b + \mathfrak{a}$ is a unit in $R/\mathfrak{a}$. That is, $R/\mathfrak{a}$ is a field.

Conversely, let $R/\mathfrak{a}$ be a field. Consider any ideal $\mathfrak{b}$ of R with $\mathfrak{a} \subsetneq \mathfrak{b} \subseteq R$. Choose any $b \in \mathfrak{b} \setminus \mathfrak{a}$. Then $b + \mathfrak{a} \neq 0 + \mathfrak{a}$. By hypothesis, there exists $c \in R$ such that $(b + \mathfrak{a})(c + \mathfrak{a}) = 1 + \mathfrak{a}$, that is, $bc - 1 \in \mathfrak{a} \subseteq \mathfrak{b}$. Hence $1 \in \mathfrak{b}$, that is, $\mathfrak{b} = R$. ■

The last proposition in conjunction with Corollary 2.1 indicates:

Corollary 2.2 Maximal ideals are prime. ■

Corollary 2.3 For every $p \in \mathbb{P}$, the quotient ring $\mathbb{Z}_p = \mathbb{Z}/p\mathbb{Z}$ is a field. In particular, $p\mathbb{Z}$ is a maximal ideal of $\mathbb{Z}$.

Proof Since $p\mathbb{Z}$ is a prime ideal of $\mathbb{Z}$, $\mathbb{Z}_p$ is an integral domain. But $\#\mathbb{Z}_p = p$ is finite, so by Exercise 2.25 $\mathbb{Z}_p$ is a field. ■

2.4.3 Homomorphisms

Recall how we have defined homomorphisms of groups. In a similar manner, we define homomorphisms of rings. A ring homomorphism is a map from one ring to another, which respects addition, multiplication and the identity element. More precisely:

Definition 2.21 Let R and S be rings. A map $f : R \to S$ is called a (ring) *homomorphism*, if $f(a+b) = f(a) + f(b)$ and $f(ab) = f(a)f(b)$ for all $a, b \in R$ and if $f(1) = 1$. A homomorphism $f : R \to S$ is called an *isomorphism*, if there exists a homomorphism $g : S \to R$ such

that $g \circ f = \mathrm{id}_R$ and $f \circ g = \mathrm{id}_S$. As in the case of groups, bijectivity of f as a function is both necessary and sufficient for a homomorphism $f : R \to S$ to be an isomorphism. If $f : R \to S$ is an isomorphism, we write $R \cong S$ and say that R is *isomorphic* to S or that R and S are *isomorphic*.

A homomorphism $f : R \to R$ is called an *endomorphism* of R. An *automorphism* is a bijective endomorphism. □

Example 2.9

(1) For any ring extension $R \subseteq S$, the canonical inclusion $a \mapsto a$ is a homomorphism from $R \to S$. In particular, the identity map on any ring is an automorphism.

(2) Let R be a ring and $\mathfrak{a}$ an ideal of R. The canonical surjection $R \to R/\mathfrak{a}$ that takes $a \mapsto a + \mathfrak{a}$ is a ring homomorphism.

(3) Let R be a ring and let $a \in R$. The map $R[X] \to R$ that takes $f(X) \mapsto f(a)$ is a ring homomorphism and is called the *substitution homomorphism*.

(4) The map $\mathbb{Z} \to \mathbb{Z}$ taking $n \mapsto -n$ is not a ring homomorphism, since it maps 1 to -1 (and does not satisfy $f(ab) = f(a)f(b)$ for all $a, b \in \mathbb{Z}$).

(5) The map $\mathbb{C} \to \mathbb{C}$ that maps $z = a + \mathrm{i}b$ to its conjugate $\bar{z} = a - \mathrm{i}b$ is an automorphism of the field $\mathbb{C}$. □

Proposition 2.10 Let $f : R \to S$ be a ring homomorphism.
(1) If $a \in R$ is a unit, then $f(a)$ is a unit in S and $f(a^{-1}) = (f(a))^{-1}$.
(2) Let $\mathfrak{b}$ be an ideal in S. Then $\mathfrak{a} := f^{-1}(\mathfrak{b}) := \{a \in R \mid f(a) \in \mathfrak{b}\}$ is an ideal in R. If $\mathfrak{b}$ is prime, then $\mathfrak{a}$ is also prime.

Proof

(1) If $ab = 1$, then $f(a)f(b) = f(ab) = f(1) = 1$.

(2) For $r \in R$, $a, a' \in \mathfrak{a}$ and $b, b' \in \mathfrak{b}$ with $f(a) = b$ and $f(a') = b'$, we have $f(a - a') = b - b' \in \mathfrak{b}$ and $f(ra) = f(r)b \in \mathfrak{b}$. Thus $\mathfrak{a}$ is an ideal of R. If $cc' \in \mathfrak{a}$, then $f(cc') = f(c)f(c') \in \mathfrak{b}$. If $\mathfrak{b}$ is prime (in which case $\mathfrak{a}$ and $\mathfrak{b}$ are proper ideals of R and S respectively), then $f(c) \in \mathfrak{b}$ or $f(c') \in \mathfrak{b}$. But then $c \in \mathfrak{a}$ or $c' \in \mathfrak{a}$. ■

The ideal $\mathfrak{a}$ of the above proposition is called the *contraction* of $\mathfrak{b}$ and is often denoted by $\mathfrak{b}^c$. If $R \subseteq S$ and f is the inclusion homomorphism, then $\mathfrak{a} = f^{-1}(\mathfrak{b}) = \mathfrak{b}^c = R \cap \mathfrak{b}$.

Definition 2.22 Let $f : R \to S$ be a ring homomorphism. The set $\{a \in R \mid f(a) = 0\}$ is called the *kernel* of f and is denoted by $\mathrm{Ker}\, f$. The set $\{b \in S \mid b = f(a) \text{ for some } a \in R\}$ is called the *image* of f and is denoted by $f(R)$ or $\mathrm{Im}\, f$. □

Theorem 2.9 **Isomorphism theorem** With the notations of the last definition, $\mathrm{Ker}\, f$ is an ideal of R, $\mathrm{Im}\, f$ is a subring of S and $R/\mathrm{Ker}\, f \cong \mathrm{Im}\, f$.

Proof Consider the map $\varphi : R/\mathrm{Ker}\, f \to \mathrm{Im}\, f$ that takes $a + \mathrm{Ker}\, f \mapsto f(a)$. It is easy to verify that φ is a well-defined ring homomorphism and is bijective. The details are left to the reader. Also see Theorem 2.3. ■

Definition 2.23 Two ideals $\mathfrak{a}$ and $\mathfrak{b}$ of a ring R are called *relatively prime* or *coprime* if $\mathfrak{a} + \mathfrak{b} = R$, that is, if there exist $a \in \mathfrak{a}$ and $b \in \mathfrak{b}$ with $a + b = 1$. □

Theorem 2.10 **Chinese remainder theorem** (CRT) Let R be a ring and $n \in \mathbb{N}$. Let $\mathfrak{a}_1, \ldots, \mathfrak{a}_n$ be ideals in R such that for all i, j, $i \neq j$, the ideals $\mathfrak{a}_i$ and $\mathfrak{a}_j$ are relatively prime. Then $R/(\mathfrak{a}_1 \cap \cdots \cap \mathfrak{a}_n)$ is isomorphic to the direct product $R/\mathfrak{a}_1 \times \cdots \times R/\mathfrak{a}_n$.

Proof The assertion is obvious for $n = 1$. So assume that $n \geqslant 2$ and define the map $\varphi : R/(\mathfrak{a}_1 \cap \cdots \cap \mathfrak{a}_n) \to R/\mathfrak{a}_1 \times \cdots \times R/\mathfrak{a}_n$ by $a + (\mathfrak{a}_1 \cap \cdots \cap \mathfrak{a}_n) \mapsto (a + \mathfrak{a}_1, \ldots, a + \mathfrak{a}_n)$ for all $a \in R$. Since $\mathfrak{a}_1 \cap \cdots \cap \mathfrak{a}_n \subseteq \mathfrak{a}_i$ for all i, the map is well-defined. It is easy to see that φ is a ring homomorphism. In order to show that φ is injective, we let $\varphi(a + (\mathfrak{a}_1 \cap \cdots \cap \mathfrak{a}_n)) = 0$. This means that $a + \mathfrak{a}_i = 0$, that is, $a \in \mathfrak{a}_i$ for all i. Then $a \in \mathfrak{a}_1 \cap \cdots \cap \mathfrak{a}_n$, that is, $a + (\mathfrak{a}_1 \cap \cdots \cap \mathfrak{a}_n) = 0$. The trickier part is to prove that φ is surjective. Let $a_1, \ldots, a_n \in R$. Let us consider the ideal $\mathfrak{b}_i := \mathfrak{a}_1 \cap \cdots \cap \mathfrak{a}_{i-1} \cap \mathfrak{a}_{i+1} \cap \cdots \cap \mathfrak{a}_n$ for each i. For a given i, there exist for each $j \neq i$ elements $\alpha_j \in \mathfrak{a}_i$ and $\beta_j \in \mathfrak{a}_j$ with $\alpha_j + \beta_j = 1$. Multiplying these equations shows that we have a $\gamma_i \in \mathfrak{a}_i$ such that $\gamma_i + \delta_i = 1$, where $\delta_i := \beta_1 \cdots \beta_{i-1}\beta_{i+1} \cdots \beta_n \in \mathfrak{b}_i$. (This shows that $\mathfrak{a}_i + \mathfrak{b}_i = R$ for all i.) Now consider the element $a := \sum_{i=1}^{n} \delta_i a_i$. It follows that $a \equiv a_i \pmod{\mathfrak{a}_i}$ for all i, that is, $\varphi(a + (\mathfrak{a}_1 \cap \cdots \cap \mathfrak{a}_n)) = (a_1 + \mathfrak{a}_1, \ldots, a_n + \mathfrak{a}_n)$. ■

In Section 2.5, we will see an interesting application of this theorem. Notice that the injectivity of φ in the last proof does not require the coprimality of $\mathfrak{a}_1, \ldots, \mathfrak{a}_n$; the surjectivity of φ requires this condition.

2.4.4 Factorization in Rings

Now we introduce the concept of divisibility in a ring. We also discuss about an important type of rings known as unique factorization domains. This study is a natural generalization of that of the rings $\mathbb{Z}$ and $K[X]$, K a field.

Definition 2.24 Let R be a ring, $a, b \in R$ and $0 \neq p \in R \setminus R^*$. Also let K be a field.

(1) We say that a *divides* b and write $a|b$, if there exists an element $c \in R$ such that $b = ac$. If a does not divide b, we write $a \not| b$. In $\mathbb{Z}$, for example, $-31|899$, since $899 = (-31) \cdot (-29)$. By this definition, any element $a \in R$ divides 0, whereas 0 divides no element other than 0.

(2) It is easy to see that $a|b$ and $b|a$ if and only if $b = ca$ for some unit $c \in R$. In that case, we say that a and b are *associates* of each other. The relation of *being associate* is an equivalence relation on R (or $R \setminus \{0\}$), as can be easily verified. The only associates of $a \in \mathbb{Z}$, $a \neq 0$, are $\pm a$, since ± 1 are the only units in $\mathbb{Z}$. Two non-zero polynomials f and g of $K[X]$ are associates if and only if $f = \alpha g$ for some $\alpha \in K \setminus \{0\}$.

(3) A non-zero non-unit $p \in R$ is called a *prime*, if $p|ab$ implies either $p|a$ or $p|b$. One can check easily that p is prime if and only if the principal ideal $\langle p \rangle = pR$ is a prime ideal.

(4) A non-zero non-unit $p \in R$ is called *irreducible*, if $p = ab$ implies either a or b is a unit. □

Note that for $\mathbb{Z}$ the concepts of prime and irreducible elements are the same. This is indeed true for any PID (Proposition 2.12). Thus our conventional definition of a prime integer $p > 0$ as one which has only 1 and p as (positive) divisors tallies with the definition of irreducible elements above. For the ring $K[X]$, on the other hand, it is more customary to talk about irreducible polynomials instead of prime polynomials; they are the same thing anyway.

Proposition 2.11 Let R be an integral domain and $p \in R$ a prime. Then p is irreducible.

Proof Let $p = ab$. Then $p|(ab)$, so that by hypothesis $p|a$ or $p|b$. If $p|a$, then $a = up$ for some $u \in R$. Hence $p = ab = upb$, that is, $(1 - ub)p = 0$. Since R is an integral domain and $p \neq 0$, we have $1 - ub = 0$, that is, $ub = 1$, that is, b is a unit. Similarly, $p|b$ implies a is a unit. ■

Proposition 2.12 Let R be a PID. An element $p \in R$ is prime if and only if p is irreducible.

Proof [if] Let p be irreducible, but not prime. Then there are $a, b \in R$ such that $a \notin \langle p \rangle$ and $b \notin \langle p \rangle$, but $ab \in \langle p \rangle$. Consider the ideal $\mathfrak{a} = \langle \alpha \rangle = \langle p \rangle + \langle a \rangle$. Since $p \in \langle \alpha \rangle$, we have $p = c\alpha$ for some $c \in R$. By hypothesis, p is irreducible, so that either c or α is a unit. If c is a unit, $\langle p \rangle = \langle \alpha \rangle = \langle p \rangle + \langle a \rangle$, that is, $a \in \langle p \rangle$, a contradiction. So α is a unit. Then $\langle p \rangle + \langle a \rangle = R$ which implies that there are elements $u, v \in R$ such that $up + va = 1$. Similarly, there are elements $u', v' \in R$ such that $u'p + v'b = 1$. Multiplying these two equations gives $(uu'p + uv'b + u'va)p + (vv')ab = 1$. Now $ab \in \langle p \rangle$, so that $ab = wp$ for some $w \in R$. But then $(uu'p + uv'b + u'va + vv'w)p = 1$, which shows that p is a unit, a contradiction.
[only if] Immediate from Proposition 2.11. ■

Definition 2.25 An integral domain R is called a *unique factorization domain* or a *UFD* in short, if every non-zero element $a \in R$ can be written as a product $a = up_1 \dots p_r$, where $u \in R^*$, $r \in \mathbb{N}_0$ and $p_1, \dots, p_r$ are prime elements (not necessarily distinct) of R. Moreover, such a factorization is unique up to permutation of the primes $p_1, \dots, p_r$ and up to multiplication of the primes by units. This factorization can also be written as $a = uq_1^{\alpha_1} \dots q_s^{\alpha_s}$, where $u \in R^*$, $s \in \mathbb{N}_0$, $q_1, \dots, q_s$ are pairwise non-associate primes and $\alpha_i > 0$ for $i = 1, \dots, s$. Some authors also use the term *factorial ring* or *factorial domain* in order to describe a UFD.

If $p \in R$ is a prime and $a \in R$, $a \neq 0$, then the *multiplicity* of p in a is the non-negative integer v such that $p^v | a$, but $p^{v+1} \nmid a$. This integer v is denoted by $\mathrm{v}_p(a)$. It is clear form the definition that for every $a \in R$, $a \neq 0$, there exist only finitely many non-associate primes p for which $\mathrm{v}_p(a) > 0$. □

Proposition 2.13 Let R be a UFD. An element $p \in R$ is prime if and only if p is irreducible.

Proof The *only if* part is immediate from Proposition 2.11. For proving the *if* part, let $p = up_1 \cdots p_r$ ($u \in R^*$ and p_i primes in R) be irreducible. If $r = 0$, p is a unit, a contradiction. If $r > 1$, then p can be written as the product of two non-units $up_1 \cdots p_{r-1}$ and p_r, again a contradiction. So $r = 1$. ■

A classical example of an integral domain that is not a UFD is $\mathbb{Z}[\sqrt{-5}] := \{a + b\sqrt{-5} \mid a, b \in \mathbb{Z}\}$. In this ring, we have two essentially different factorizations $6 =$

$2 \times 3 = (1+\sqrt{-5})(1-\sqrt{-5})$ of 6 into irreducible elements. The failure of irreducible elements to be primes in such rings is a serious thing to patch up!

Theorem 2.11 A PID is a UFD

Proof Let R be a PID and $a \in R \setminus \{0\}$. We show that a has a factorization of the form $a = up_1 \dots p_r$, where u is a unit and $p_1, \dots, p_r$ are prime elements of R. If a is a unit, we are done. So assume that $a =: a_0$ is a non-unit and let $\mathfrak{a}_0 = \langle a_0 \rangle$. Since $\mathfrak{a}_0 \neq \langle 1 \rangle$, there is a maximal ideal $\mathfrak{m}_1 = \langle p_1 \rangle$ containing $\mathfrak{a}_0$ (Exercise 2.23). Then p_1 is a prime that divides a_0. Let $a_0 = a_1 p_1$. We have $\mathfrak{a}_0 \subseteq \mathfrak{a}_1 := \langle a_1 \rangle$. If $\mathfrak{a}_1$ is the unit ideal, we are done. Otherwise we choose as before a prime p_2 dividing a_1 and with $a_1 = a_2 p_2$ get the ideal $\mathfrak{a}_2 := \langle a_2 \rangle$ properly containing $\mathfrak{a}_1$. Repeating this process we can generate a strictly ascending chain $\mathfrak{a}_0 \subsetneqq \mathfrak{a}_1 \subsetneqq \mathfrak{a}_2 \subsetneqq \cdots$ of ideals of R. Since R is a PID and hence Noetherian, this process must stop after finitely many steps (Exercise 2.33). ■

The converse of the above theorem is not necessarily true. For example, the polynomial ring $K[X_1, \dots, X_n]$ over a field K is a UFD for every $n \in \mathbb{N}$, but not a PID for $n \geqslant 2$.

Divisibility in a UFD can be rephrased in terms of prime factorizations. Let R be a UFD and let the non-zero elements $a, b \in R$ have the prime factorizations $a = up_1^{\alpha_1} \dots p_r^{\alpha_r}$ and $b = u' p_1^{\beta_1} \dots p_r^{\beta_r}$ with units u, u', pairwise non-associate primes $p_1, \dots, p_r$ and with $\alpha_i \geqslant 0$ and $\beta_i \geqslant 0$. Then $a|b$ if and only if $\alpha_i \leqslant \beta_i$ for all $i = 1, \dots, r$. This notion leads to the following definitions.

Definition 2.26 Let R be a UFD and let $a, b \in R \setminus \{0\}$ have prime factorizations as in the last paragraph. Any associate of $p_1^{\gamma_1} \dots p_r^{\gamma_r}$, $\gamma_i := \min(\alpha_i, \beta_i)$, is called a *greatest common divisor* of a and b and is denoted by $\gcd(a, b)$. Clearly, $\gcd(a, b)$ is unique up to multiplication by units of R. Similarly, any associate of $p_1^{\delta_1} \dots p_r^{\delta_r}$, $\delta_i := \max(\alpha_i, \beta_i)$, is called a *least common multiple* of a and b and is denoted by $\operatorname{lcm}(a, b)$. $\operatorname{lcm}(a, b)$ is again unique up to multiplication by units of R. The gcd of $a \neq 0$ and 0 is taken to be an associate of a, whereas $\gcd(0, 0)$ is undefined. On the other hand, $\operatorname{lcm}(a, 0)$ is defined to be 0 for any $a \in R$. □

It is clear that these definitions of gcd and lcm can be readily generalized for any arbitrary *finite* number of elements.

Corollary 2.4 Let R be a UFD and $a, b \in R$ not both zero. Then $\gcd(a, b) \cdot \operatorname{lcm}(a, b)$ is an associate of ab.

Proof Immediate from the definitions. ■

Corollary 2.5 Let R be a UFD and $a, b, c \in R$ with $a|bc$. If $\gcd(a, c) = 1$, then $a|b$.

Proof Consider the prime factorizations of a, b and c. ■

For a PID, the gcd and lcm have equivalent characterizations.

Proposition 2.14 Let R be a PID and a, b be non-zero elements of R. Let d be a gcd of a and b. Then $\langle d \rangle = \langle a \rangle + \langle b \rangle$. If f is an lcm of a and b, then $\langle f \rangle = \langle a \rangle \cap \langle b \rangle$.

Proof Let $\langle a \rangle + \langle b \rangle = \langle c \rangle$. We show that c and d are associates. There exist $u, v \in R$ such that $ua + vb = c$. Since $d|a$ and $d|b$, we have $d|c$. On the other hand, $a \in \langle c \rangle$, so that

$c|a$. Similarly $c|b$. Considering the prime factorizations of a and b one can then readily verify that $c|d$. The proof for the second part is similar and is left to the reader. ■

A direct corollary to the last proposition is the following.

Corollary 2.6 Let R be a PID, $a, b \in R$ (not both zero) and d a gcd of a and b. Then there are elements $u, v \in R$ such that $ua + vb = d$. In particular, the ideals $\langle a \rangle$ and $\langle b \rangle$ are relatively prime if and only if $\gcd(a, b)$ is a unit. In that case, we also say that the elements a and b are *relatively prime* or *coprime*. ■

This completes our short survey of factorization in rings. Note that $\mathbb{Z}$ and $K[X]$ (for a field K) are PID and hence UFD. Thus all the results we have proved in this section apply equally well to both these rings. It is because of this (and not of a mere coincidence) that these two rings enjoy many common properties. Thus our abstract treatment saves us from the duplicate effort of proving the same results once for integers (Section 2.5) and once more for polynomials (Section 2.6).

EXERCISE SET 2.4

2.21 For a *non-zero* ring R, prove the following assertions:
(a) A unit of R is not a zero-divisor.
(b) The product of two units of R is again a unit.
(c) The product of two non-units of R is again a non-unit.
(d) The element 0 is not a unit in R.
(e) The element 1 is always a unit in R.
(f) If a is a unit and $ab = ac$, then $b = c$.
Let K be a field. What are the units in the polynomial ring $K[X]$? In $K[X_1, \ldots, X_n]$? In the ring $K(X)$ of rational functions? In $K(X_1, \ldots, X_n)$?

2.22 Binomial theorem Let R be a ring, $a, b \in R$ and $n \in \mathbb{N}$. Show that

$$(a+b)^n = \sum_{r=0}^{n} \binom{n}{r} a^{n-r} b^r \,,$$

where

$$\binom{n}{r} = \frac{n!}{r!(n-r)!} = \frac{n(n-1)\cdots(n-r+1)}{r!}$$

are the binomial coefficients.

2.23 Show that every non-zero ring has a maximal (and hence prime) ideal. More generally, show that every non-unit ideal of a non-zero ring is contained in a maximal ideal. **[H]**

2.24 Let R be a ring.
(a) Show that the set of all nilpotent elements of R is an ideal of R. This ideal is usually denoted by $\mathfrak{n}_R$ and is called the *nilradical* of R.
(b) Show that the quotient ring $R/\mathfrak{n}_R$ has no non-zero nilpotent elements. (The ring $R/\mathfrak{n}_R$ is called the *reduction* of R and is often written as R_{red}. If $\mathfrak{n}_R = 0$, then we say that R is *reduced*. Thus $R_{\mathrm{red}} = R/\mathfrak{n}_R$ is always reduced.)

(c) Show that the nilradical of R is the intersection of the prime ideals of R. **[H]**

2.25 Show that a finite integral domain R is a field. **[H]**

2.26 Let R be a ring of characteristic 0. Show that:
(a) R contains infinitely many elements.
(b) If R is an integral domain, then R contains as subring an isomorphic copy of $\mathbb{Z}$.
(c) If R is a field, then R contains as subfield an isomorphic copy of $\mathbb{Q}$.

2.27 Let $f : R \to S$ be a ring-homomorphism and let $\mathfrak{a}$ and $\mathfrak{b}$ be ideals in R and S respectively. Find examples to corroborate the following statements.
(a) Let $a \in R$ be such that $f(a)$ is a unit in S. Then a need not be a unit in R.
(b) The set $f(\mathfrak{a}) = \{b \in S \mid b = f(a) \text{ for some } a \in R\}$ need not be an ideal of S.
(c) If $\mathfrak{a} = f^{-1}(\mathfrak{b})$ and if $\mathfrak{b}$ is maximal, then $\mathfrak{a}$ need not be maximal.

2.28 Let K be a field.
(a) Show that a homomorphism from K to any non-zero ring is injective.
(b) Let L be another field and let $f : K \to L$ and $g : L \to K$ be homomorphisms such that $g \circ f = \mathrm{id}_K$. Show that f and g are isomorphisms.

2.29 **(a)** Show that a ring R is an integral domain if and only if 0 is a prime ideal of R.
(b) Give an example of a reduced ring that is *not* an integral domain. (Note that an integral domain is always reduced.)

2.30 Let R be a ring and let $\mathfrak{a}$ and $\mathfrak{b}$ be ideals of R with $\mathfrak{a} \subseteq \mathfrak{b}$. Show that $\mathfrak{b}/\mathfrak{a} := \{b+\mathfrak{a} \mid b \in \mathfrak{b}\}$ is an ideal of $R/\mathfrak{a}$ and that $(R/\mathfrak{a})/(\mathfrak{b}/\mathfrak{a}) \cong R/\mathfrak{b}$. **[H]**

2.31 An integral domain R is called a *Euclidean domain* (ED) if there is a map $\nu : R\setminus\{0\} \to \mathbb{N}_0$ satisfying the following two conditions:

(1) $\nu(a) \leqslant \nu(ab)$ for all $a, b \in R \setminus \{0\}$.
(2) For every $a, b \in R$ with $b \neq 0$, there exist (not necessarily unique) $q, r \in R$ such that $a = qb + r$ with $r = 0$ or $\nu(r) < \nu(b)$.

Show that:
(a) $\mathbb{Z}$ is a Euclidean domain with $\nu(a) = |a|$ for $a \neq 0$.
(b) The polynomial ring $K[X]$ over a field K is a Euclidean domain with $\nu(a) = \deg a$ for $a \neq 0$.
(c) For $d = -2, -1, 2, 3$, the ring

$$\mathbb{Z}[\sqrt{d}] := \{a + b\sqrt{d} \mid a, b \in \mathbb{Z}\}$$

is a Euclidean domain with $\nu(a + b\sqrt{d}) = |a^2 - db^2|$, $a, b \in \mathbb{Z}$, not both 0.
(d) A Euclidean domain is a PID (and hence a UFD).

2.32 Let R be a ring and $\mathfrak{a} \subseteq R$ an ideal. Consider the set

$$\sqrt{\mathfrak{a}} := \{a \in R \mid a^n \in \mathfrak{a} \text{ for some } n \in \mathbb{N}\}.$$

Show that $\sqrt{\mathfrak{a}}$ is an ideal of R. It is called the *radical* or *root* of $\mathfrak{a}$. If $\sqrt{\mathfrak{a}} = \mathfrak{a}$, then $\mathfrak{a}$ is called a *radical* or a *root ideal*. For arbitrary ideals $\mathfrak{a}$ and $\mathfrak{b}$ of R, prove the following assertions.

(a) $\mathfrak{a} \subseteq \sqrt{\mathfrak{a}}$.
(b) $\sqrt{\sqrt{\mathfrak{a}}} = \sqrt{\mathfrak{a}}$.
(c) If $\mathfrak{a} \subseteq \mathfrak{b}$, then $\sqrt{\mathfrak{a}} \subseteq \sqrt{\mathfrak{b}}$.
(d) If $\mathfrak{a}$ is a prime ideal, then $\sqrt{\mathfrak{a}} = \mathfrak{a}$.
(e) $\sqrt{\mathfrak{a}} = R$ if and only if $\mathfrak{a} = R$.
(f) $\sqrt{\mathfrak{a} + \mathfrak{b}} = \sqrt{\sqrt{\mathfrak{a}} + \sqrt{\mathfrak{b}}}$.
(g) $\sqrt{\mathfrak{a} \cap \mathfrak{b}} = \sqrt{\mathfrak{a}} \cap \sqrt{\mathfrak{b}}$.
(h) The nilradical $\mathfrak{n}_R = \sqrt{0}$.

2.33 Let R be a ring. An *ascending chain* of ideals $\mathfrak{a}_1, \mathfrak{a}_2, \ldots$ of R is a sequence $\mathfrak{a}_1 \subseteq \mathfrak{a}_2 \subseteq \cdots$. The ascending chain is called *stationary*, if there is some $n_0 \in \mathbb{N}$ such that $\mathfrak{a}_{n+1} = \mathfrak{a}_n$ for all $n \geqslant n_0$. Show that the following conditions are equivalent. **[H]**
(1) R is Noetherian (that is, every ideal of R is finitely generated).
(2) Every ascending chain of ideals in R is stationary.
(3) Every non-empty set of ideals of R has a maximal element.

2.34 **(a)** Let R be an integral domain. Define the set $S := \{(a, b) \mid a \in R, b \in R \setminus \{0\}\}$. Define a relation $\sim$ on S as $(a, b) \sim (c, d)$ if and only if $ad = bc$. Show that $\sim$ is an equivalence relation on S. Let us denote the equivalence class of $(a, b) \in S$ by a/b and the set of all equivalence classes of S under $\sim$ by K.
(b) Now define $(a/b)+(c/d) := (ad+bc)/(bd)$ and $(a/b)\cdot(c/d) := (ac)/(bd)$. Show that these definitions make K a field. This field is called the *quotient field* of R and is denoted as $\mathrm{Q}(R)$. This process resembles the formation of rational numbers from the integers. Indeed, $\mathbb{Q} = \mathrm{Q}(\mathbb{Z})$.

2.5 Integers

The set $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$ of integers is the main object of study in this section. We use many results from previous sections to derive properties of integers. Recall that $\mathbb{Z}$ is a PID and hence a UFD.

2.5.1 Divisibility

The notions of divisibility, prime and relatively prime integers, gcd and lcm of integers are essentially the same as discussed in connection with a PID or a UFD. We avoid repeating the definitions here, but concentrate on other useful properties of integers, not covered so far. We only mention that whenever we talk about a prime integer, or the gcd or lcm of two or more integers, we will usually refer to a non-negative integer. This convention makes primes, gcds and lcms unique.

Theorem 2.12 There are infinitely many prime integers.

Proof Let $n \in \mathbb{N}$ be arbitrary and let $p_1, p_2, \ldots, p_n$ be n distinct primes. The (non-zero non-unit) integer $q := p_1 p_2 \cdots p_n + 1$ is divisible by neither of $p_1, \ldots, p_n$ and hence must have a prime divisor p_{n+1} different from $p_1, \ldots, p_n$. The result then follows by induction on n (and the fact that the set of primes is non-empty). ■

Theorem 2.13 For an integer a and an integer $b \neq 0$, there exist unique integers q and r such that $a = qb + r$ with $0 \leqslant r < |b|$.

Proof Call the smallest non-negative element in the set $\{a - cb \mid c \in \mathbb{Z}\}$ to be r and the corresponding value of c to be q. Then these integers q and r satisfy the desired properties. To prove the uniqueness let $a = q_1 b + r_1 = q_2 b + r_2$, where $0 \leqslant r_1 < |b|$ and $0 \leqslant r_2 < |b|$. But then $(q_2 - q_1)b = r_1 - r_2$ with $-|b| < r_1 - r_2 < |b|$. Since $b|(r_1 - r_2)$, we must then have $r_1 - r_2 = 0$, that is, $r_1 = r_2$, which, in turn, implies that $q_1 = q_2$. ■

The integers q and r in the above theorem are respectively called the *quotient* and the *remainder* of *Euclidean division* of a by b and are denoted respectively by $a \operatorname{quot} b$ and $a \operatorname{rem} b$. Do not confuse Euclidean division with the division (that is, the inverse of multiplication) of the ring $\mathbb{Z}$. Euclidean division is the basis of the *Euclidean gcd* algorithm. More specifically:

Proposition 2.15 For integers a, b with $b \neq 0$, let r be the remainder of Euclidean division of a by b. Then $\gcd(a, b) = \gcd(b, r)$.

Proof Clearly, $\langle a \rangle + \langle b \rangle = \langle r \rangle + \langle b \rangle$. Now use Proposition 2.14. ■

Proposition 2.16 Let a and b be two integers, not both zero, and let d be the (positive) gcd of a and b. Then there are integers u and v such that $d = ua + vb$. (Such an equality is called a *Bézout relation*.) Furthermore, if a and b are both non-zero and $(|a|, |b|) \neq (1, 1)$, then u and v can be so chosen that $|u| < |b|$ and $|v| < |a|$.

Proof The existence of u and v follows immediately from Proposition 2.14. If $a = qb$, then $u = 0$ and $v = 1$ is a suitable choice. So assume that $a \nmid b$ and $b \nmid a$, in which case $d < |a|$ and $d < |b|$. We may assume, without loss of generality, that a and b are positive. First note that if (u, v) satisfies the Bézout relation, then for any $k \in \mathbb{Z}$ the pair $(u + kb, v - ka)$ also satisfies the same relation. So we may replace v by its remainder of Euclidean division by a and may assume $|v| < a$. But then $|ua| - b < |ua| - d \leqslant |ua - d| = |vb| \leqslant (a - 1)b$, which implies $|u| < b$. ■

The notions of the gcd and of the Bézout relation can be generalized to any finite number of integers $a_1, \ldots, a_n$ as

$$\gcd(a_1, \ldots, a_n) = \gcd(\cdots(\gcd(\gcd(a_1, a_2), a_3)\cdots), a_n) = u_1 a_1 + \cdots + u_n a_n$$

for some integers $u_1, \ldots, u_n$ (provided that all the gcds mentioned are defined).

2.5.2 Congruences

Since $\mathbb{Z}$ is a PID, congruence modulo a non-zero ideal of $\mathbb{Z}$ can be rephrased in terms of congruence modulo a positive integer as follows.

Definition 2.27 Let $n \in \mathbb{N}$. Two integers a and b are said to be *congruent modulo* n, denoted $a \equiv b \pmod{n}$, if $n|(a-b)$, that is, if the remainders of Euclidean divisions of a and b by n are the same. In terms of ideals, this is the same as $a \equiv b \pmod{\langle n \rangle}$ (See Definition 2.20). Congruence is an equivalence relation on $\mathbb{Z}$, the equivalence classes being the cosets of the ideal $n\mathbb{Z} = \langle n \rangle$ of $\mathbb{Z}$. □

By an abuse of notation, we often denote the equivalence class $[a]$ of $a \in \mathbb{Z}$ simply by a. The following are some basic properties of congruent integers.

Proposition 2.17 Let $n \in \mathbb{N}$, $a \equiv b \pmod{n}$ and $c \equiv d \pmod{n}$. Then:
(1) $a \pm c \equiv b \pm d \pmod{n}$.
(2) $ac \equiv bd \pmod{n}$.
(3) For any polynomial $f(X) \in \mathbb{Z}[X]$, we have $f(a) \equiv f(b) \pmod{n}$.
(4) If $n'|n$, then $a \equiv b \pmod{n'}$.
(5) If $m|a$ and $m|b$, then $a/m \equiv b/m \pmod{n/\gcd(n,m)}$.

Proof (1) and (2) follow from the consideration of the quotient ring $\mathbb{Z}/n\mathbb{Z}$. (3) follows from repeated applications of (1) and (2). For the proof of (4), consider $a - b = kn$ and $n = k'n'$ for $k, k' \in \mathbb{Z}$. For proving (5), take $a - b = kn = lm$. Then $m/\gcd(n,m)$ divides $k(n/\gcd(n,m))$. Since $m/\gcd(n,m)$ and $n/\gcd(n,m)$ are coprime, by Corollary 2.5 $l' := k/(m/\gcd(n,m))$ is an integer and we have $a/m - b/m = l = kn/m = l'(n/\gcd(n,m))$. ■

Let $n_1, \ldots, n_r \in \mathbb{N}$ with $\gcd(n_i, n_j) = 1$ for $i \neq j$. Then $\operatorname{lcm}(n_1, \ldots, n_r) = n_1 \cdots n_r$, and by the Chinese remainder theorem (Theorem 2.10), we have

$$\mathbb{Z}_{n_1} \times \cdots \times \mathbb{Z}_{n_r} \cong \mathbb{Z}_{n_1 \cdots n_r}.$$

This implies that, given integers $a_1, \ldots, a_r$, there exists an integer x unique modulo $n_1 \cdots n_r$ such that x satisfies the following congruences simultaneously:

$$\begin{aligned} x &\equiv a_1 \pmod{n_1} \\ x &\equiv a_2 \pmod{n_2} \\ &\vdots \\ x &\equiv a_r \pmod{n_r} \end{aligned}$$

We now give a procedure for constructing the integer x explicitly. Define $N := n_1 \cdots n_r$ and $N_i := N/n_i$ for $1 \leqslant i \leqslant r$. Then for each i we have $\gcd(n_i, N_i) = 1$ and, therefore, there are integers u_i and v_i with $u_i n_i + v_i N_i = 1$. Then $x \equiv \sum_{i=1}^{r} a_i v_i N_i \pmod{N}$ is the desired solution.

Let $n \in \mathbb{N}$. We now study the multiplicative group $\mathbb{Z}_n^*$ of the ring $\mathbb{Z}_n$. We say that an integer $a \in \mathbb{Z}$ has a *multiplicative inverse modulo* n, if $a \in \mathbb{Z}_n^*$, or, equivalently, if there is an integer b with $ab \equiv 1 \pmod{n}$. The following proposition is an important characterization of the elements of $\mathbb{Z}_n^*$.

Proposition 2.18 (The equivalence class of) an integer a belongs to $\mathbb{Z}_n^*$ if and only if $\gcd(a, n) = 1$.

Proof [if] By Proposition 2.16, there exist integers u and v such that $ua + vn = 1$. But then $ua \equiv 1 \pmod{n}$.

[only if] For some integers u and v, we have $ua + vn = 1$, which implies that the gcd of a and n divides 1 and hence is equal to 1. ■

Definition 2.28 The cardinality of $\mathbb{Z}_n^*$ is denoted by $\phi(n)$. By Proposition 2.18, $\phi(n)$ is equal to the number of integers between 0 and $n-1$ (both inclusive), which are relatively prime to n. The function $\phi : \mathbb{N} \to \mathbb{N}$ is called *Euler's totient function.* For example, for a prime p we have $\mathbb{Z}_p^* = \{1, \ldots, p-1\}$, so $\phi(p) = p - 1$. □

The following two theorems are immediate consequences of Proposition 2.4.

Theorem 2.14 **Euler's theorem** Let $n \in \mathbb{N}$ and $a \in \mathbb{Z}$ with $\gcd(a, n) = 1$. Then

$$a^{\phi(n)} \equiv 1 \pmod{n}.$$

■

Theorem 2.15 **Fermat's little theorem** Let p be a prime and $a \in \mathbb{Z}$ with $\gcd(a, p) = 1$. Then

$$a^{p-1} \equiv 1 \pmod{p}.$$

For any integer $b \in \mathbb{Z}$, one has $b^p \equiv b \pmod{p}$. ■

Theorem 2.16 **Wilson's theorem** For every prime p, we have $(p-1)! \equiv -1 \pmod{p}$.

Proof The result holds for $p = 2$. So assume that p is an odd prime. Since $\mathbb{Z}_p$ is a field, Fermat's little theorem gives the factorization

$$X^{p-1} - 1 \equiv (X-1)(X-2)\cdots(X-(p-1)) \pmod{p}. \tag{2.1}$$

Looking at the constant terms in two sides proves Wilson's theorem. ■

The structure of the group $\mathbb{Z}_p^*$, $p \in \mathbb{P}$, can be easily deduced from Fermat's little theorem. This gives us the following important result.

Proposition 2.19 For a prime p, the group $\mathbb{Z}_p^*$ is cyclic.

Proof For every divisor d of $p-1$, we have $X^{p-1}-1 = (X^d-1)f(X)$ for some $f(X) \in \mathbb{Z}[X]$ with $\deg f = p-1-d$. By Congruence 2.1, $X^{p-1}-1$ has $p-1$ roots modulo p. Since $\mathbb{Z}_p$ is a field, $f(X) \pmod{p}$ cannot have more than $p-1-d$ roots (Proposition 2.25) and it follows that X^d-1 has exactly d roots modulo p. In particular, if $d = q^e$ for some $q \in \mathbb{P}$ and $e \in \mathbb{N}$, then there exist exactly q^e elements of $\mathbb{Z}_p^*$ of orders dividing q^e and exactly q^{e-1} elements of $\mathbb{Z}_p^*$ of orders dividing q^{e-1}, that is, there are $q^e - q^{e-1} > 0$ elements of $\mathbb{Z}_p^*$ of order q^e. If $p - 1 = q_1^{e_1} \cdots q_r^{e_r}$ is the canonical prime factorization of $p-1$ (with each $e_i \geqslant 1$), by the above argument there exists an element $a_i \in \mathbb{Z}_p^*$ of order $p_i^{e_i}$ for each $i = 1, \ldots, r$. It is now easy to check that $a_1 \cdots a_r \in \mathbb{Z}_p^*$ has order $q_1^{e_1} \cdots q_r^{e_r} = p - 1$. ■

Euler's totient function plays an extremely important role in number theory (and cryptology). We now describe a method for computing it.

Lemma 2.2 If n and n' are relatively prime positive integers, then $\phi(nn') = \phi(n)\phi(n')$.

Proof If a is invertible modulo nn', then clearly it is invertible modulo both n and n'. Conversely, if $ua \equiv 1 \pmod{n}$ and $u'a' \equiv 1 \pmod{n'}$, then by the Chinese remainder theorem there are integers x and α, unique modulo nn', satisfying $x \equiv u \pmod{n}$,

$x \equiv u' \pmod{n'}$, $\alpha \equiv a \pmod{n}$ and $\alpha \equiv a' \pmod{n'}$. But then $x\alpha \equiv 1 \pmod{nn'}$. Therefore, $\mathbb{Z}^*_{nn'} \cong \mathbb{Z}^*_n \times \mathbb{Z}^*_{n'}$, whence the lemma follows. ■

Lemma 2.3 If p is a prime and $e \in \mathbb{N}$, then $\phi(p^e) = p^e - p^{e-1} = p^e(1 - 1/p)$.

Proof Integers between 0 and $p^e - 1$, which are relatively prime to p^e are precisely those that are not multiples of p. ■

Proposition 2.20 Let $n = p_1^{e_1} \cdots p_r^{e_r}$ be the prime factorization of a positive integer n with $r \in \mathbb{N}_0$, with pairwise distinct primes $p_1, \ldots, p_r$ and with $e_i > 0$. Then

$$\begin{aligned} \phi(n) &= (p_1^{e_1} - p_1^{e_1-1}) \cdots (p_r^{e_r} - p_r^{e_r-1}) \\ &= n\left(1 - \frac{1}{p_1}\right) \cdots \left(1 - \frac{1}{p_r}\right) = n \prod_{\substack{p \in \mathbb{P} \\ p|n}} \left(1 - \frac{1}{p}\right). \end{aligned}$$

Proof Immediate from Lemmas 2.2 and 2.3. ■

By Proposition 2.18, the linear congruence $ax \equiv 1 \pmod{n}$ is solvable for x if and only if $\gcd(a, n) = 1$. In such a case, the solution is unique modulo n. Now, let us concentrate on the solutions of the general *linear congruence*:

$$ax \equiv b \pmod{n}.$$

Theorem 2.17 characterizes the solutions of this congruence.

Theorem 2.17 Let $d := \gcd(a, n)$. Then the congruence $ax \equiv b \pmod{n}$ is solvable for x if and only if $d|b$. A solution of the congruence, if existent, is unique modulo n/d.

Proof [if] By Proposition 2.17, $(a/d)x \equiv b/d \pmod{n/d}$. Since $\gcd(a/d, n/d) = 1$, the congruence $(a/d)x' \equiv 1 \pmod{n/d}$ is solvable for x'. Then a solution for x is $x \equiv (b/d)x' \pmod{n/d}$.
[only if] There exists an integer k such that $ax + kn = b$. This shows that $d|b$.

To prove the uniqueness let x and x' be two integers satisfying the given congruence. But then $a(x - x') \equiv 0 \pmod{n}$, that is, $(a/d)(x - x') \equiv 0 \pmod{n/d}$, that is, $x - x' \equiv 0 \pmod{n/d}$, since $\gcd(a/d, n/d) = 1$. ■

The last theorem implies that if $d|b$, then the congruence $ax \equiv b \pmod{n}$ has d solutions modulo n. These solutions are given by $\xi + r(n/d)$, $r = 0, \ldots, d - 1$, where ξ is the solution modulo n/d of the congruence $(a/d)\xi \equiv b/d \pmod{n/d}$.

2.5.3 Quadratic Residues

In this section, we consider quadratic congruences, that is, congruences of the form $ax^2 + bx + c \equiv 0 \pmod{n}$. We start with the simple case $n = p \in \mathbb{P}$. We assume further that p is odd, so that 2 has a multiplicative inverse mod p. Since we are considering quadratic equations, we are interested only in those integers a for which $\gcd(a, p) = 1$. In that case, a also has a multiplicative inverse mod p and the above congruence can be written as $y^2 \equiv \alpha \pmod{p}$, where $y \equiv x + b(2a)^{-1} \pmod{p}$ and $\alpha \equiv b^2(4a^2)^{-1} - c(a^{-1}) \pmod{p}$. This motivates us to provide Definition 2.29.

Definition 2.29 Let p be an odd prime and a an integer with $\gcd(a, p) = 1$. We say that a is a *quadratic residue modulo* p, if the congruence $x^2 \equiv a \pmod{p}$ has a solution (for x). Otherwise we say that a is a *quadratic non-residue modulo* p. □

If a is a quadratic residue modulo an odd prime p, then the equation $x^2 \equiv a \pmod{p}$ has exactly two solutions. If ξ is one solution, the other solution is $p-\xi$. It is, therefore, evident that there are exactly $(p-1)/2$ quadratic residues and exactly $(p-1)/2$ quadratic non-residues modulo p. For example, the quadratic residues modulo $p = 11$ are $1 = 1^2 = 10^2$, $3 = 5^2 = 6^2$, $4 = 2^2 = 9^2$, $5 = 4^2 = 7^2$ and $9 = 3^2 = 8^2$. The quadratic non-residues modulo 11 are, therefore, 2, 6, 7, 8 and 10. We treat 0 neither as a quadratic residue nor as a quadratic non-residue.

Definition 2.30 Let p be an odd prime and a an integer with $\gcd(a, p) = 1$. The *Legendre symbol* $\left(\frac{a}{p}\right)$ is defined as:

$$\left(\frac{a}{p}\right) := \begin{cases} 1 & \text{if } a \text{ is a quadratic residue modulo } p, \\ -1 & \text{if } a \text{ is a quadratic non-residue modulo } p. \end{cases}$$ □

Proposition 2.21 Let p be an odd prime and a and b integers coprime to p.

(1) **Euler's criterion** $\left(\frac{a}{p}\right) \equiv a^{(p-1)/2} \pmod{p}$.

(2) $\left(\frac{ab}{p}\right) = \left(\frac{a}{p}\right)\left(\frac{b}{p}\right)$.

(3) $\left(\frac{a^2}{p}\right) = 1$, $\left(\frac{1}{p}\right) = 1$ and $\left(\frac{-1}{p}\right) = (-1)^{(p-1)/2}$.

(4) If $a \equiv b \pmod{p}$, then $\left(\frac{a}{p}\right) = \left(\frac{b}{p}\right)$. In particular, if r is the remainder of Euclidean division of a by p, then $\left(\frac{a}{p}\right) = \left(\frac{r}{p}\right)$.

Proof If a is a quadratic residue modulo p, then $a \equiv b^2 \pmod{p}$ for some integer b (coprime to p) and by Fermat's little theorem we have $a^{(p-1)/2} \equiv b^{p-1} \equiv 1 \pmod{p}$. Conversely, the polynomial $X^{p-1} - 1 = (X^{(p-1)/2} - 1)(X^{(p-1)/2} + 1)$ has $p-1$ (distinct) roots mod p (again by Fermat's little theorem). We have seen that no quadratic residues are roots of $X^{(p-1)/2} + 1$. Since $\mathbb{Z}_p$ is a field, the $(p-1)/2$ roots of $X^{(p-1)/2} - 1$ are precisely all the quadratic residues modulo p. This proves Euler's criterion. The other statements are immediate consequences of this. ■

Euler's criterion gives us a nice way to check if a given integer is a quadratic residue modulo an odd prime. While this is much faster than the brute-force strategy of enumerating all the quadratic residues, it is still not the best solution, because it involves a modular exponentiation. We can, however, employ a gcd-like procedure for a faster computation. The development of this method demands further results which are otherwise interesting in themselves as well. The first important result is known as the *law of quadratic reciprocity* (Theorem 2.18 below). Gauss was the first to prove it and he deemed the result so important that he gave eight proofs for it. At present about two hundred published proofs of this law exist in the literature. We go in the classical way, that is, the Gaussian way, because the proof, though somewhat long, is elementary.

Lemma 2.4 **Gauss** Let p be an odd prime and a an integer with $\gcd(a,p)=1$. Let us denote $t := (p-1)/2$. For an integer i, let r_i be the unique integer with $r_i \equiv ia \pmod p$ and $-t \leqslant r_i \leqslant t$. Let n be the number of i, $1 \leqslant i \leqslant t$, for which r_i is negative. Then $\left(\frac{a}{p}\right) = (-1)^n$.

Proof It is easy to check that $r_i \not\equiv \pm r_j \pmod p$ for all $i \neq j$ with $1 \leqslant i,j \leqslant t$. Thus $|r_i|$, $i = 1,\ldots,t$, are precisely (a permuted version of) the integers $1,\ldots,t$. Thus $t!\,a^t \equiv a \cdot 2a \cdot 3a \cdots ta \equiv \prod_{i=1}^t r_i \equiv (-1)^n 1 \cdot 2 \cdot 3 \cdots t \equiv (-1)^n\, t! \pmod p$. Canceling $t!$ and using Proposition 2.21(1) gives the desired result. ■

Definition 2.31 Let $x \in \mathbb{R}$. The largest integer smaller than or equal to x is called the *floor* of x and is denoted by $\lfloor x \rfloor$. Similarly, the smallest integer larger than or equal to x is called the *ceiling* of x and is denoted by $\lceil x \rceil$. □

Corollary 2.7 With the notations of Lemma 2.4 we have $n \equiv \sum_{j=1}^t \left\lfloor \frac{2ja}{p} \right\rfloor \pmod 2$. If a is odd, then $n \equiv \sum_{j=1}^t \left\lfloor \frac{ja}{p} \right\rfloor \pmod 2$. In particular, $\left(\frac{2}{p}\right) = (-1)^{\frac{p^2-1}{8}}$, that is, 2 is a quadratic residue mod p if and only if $p \equiv \pm 1 \pmod 8$.

Proof Since $\left\lfloor \frac{2ja}{p} \right\rfloor = 2\left\lfloor \frac{ja}{p} \right\rfloor + \left\lfloor 2\left(\frac{ja}{p} - \left\lfloor \frac{ja}{p} \right\rfloor\right)\right\rfloor$ is even $\iff \frac{ja}{p} - \left\lfloor \frac{ja}{p} \right\rfloor < \frac{1}{2} \iff ja - p\left\lfloor \frac{ja}{p} \right\rfloor \leqslant \frac{p-1}{2} = t \iff r_j > 0$, it follows that if $r_j > 0$, then $\left\lfloor \frac{2ja}{p} \right\rfloor$ is even, and if $r_j < 0$, then $\left\lfloor \frac{2ja}{p} \right\rfloor$ is odd. Therefore, $n \equiv \sum_{j=1}^t \left\lfloor \frac{2ja}{p} \right\rfloor \pmod 2$.

If a is odd, $p+a$ is even. Also 4 is a quadratic residue modulo p. So $\left(\frac{2}{p}\right)\left(\frac{a}{p}\right) = \left(\frac{4(p+a)/2}{p}\right) = \left(\frac{(p+a)/2}{p}\right) = (-1)^{n_1+n_2}$, where $n_1 \equiv \sum_{j=1}^t \left\lfloor \frac{ja}{p} \right\rfloor \pmod 2$ and $n_2 \equiv \sum_{j=1}^t j = t(t+1)/2 = (p^2-1)/8 \pmod 2$. Putting $a=1$ gives $\left(\frac{2}{p}\right) = (-1)^{n_2}$ and, therefore, $\left(\frac{2}{p}\right)\left(\frac{a}{p}\right) = (-1)^{n_1}\left(\frac{2}{p}\right)$, that is, $\left(\frac{a}{p}\right) = (-1)^{n_1}$. ■

Theorem 2.18 **Law of quadratic reciprocity** Let p and q be distinct odd primes. Then

$$\left(\frac{p}{q}\right)\left(\frac{q}{p}\right) = (-1)^{\frac{p-1}{2}\frac{q-1}{2}}.$$

Proof By Corollary 2.7, $\left(\frac{p}{q}\right)\left(\frac{q}{p}\right) = (-1)^{m+n}$, where $m = \sum_{j=1}^s \left\lfloor \frac{jp}{q} \right\rfloor$, $n = \sum_{j=1}^t \left\lfloor \frac{jq}{p} \right\rfloor$, $s = (q-1)/2$ and $t = (p-1)/2$. So we are done, if we can show that $m+n = st$. Consider the set $S := \{(x,y) \mid 1 \leqslant x \leqslant s, 1 \leqslant y \leqslant t\}$ of cardinality st. Now S is the disjoint union of S_1 and S_2, where $S_1 := \{(x,y) \in S \mid px > qy\}$ and $S_2 := \{(x,y) \in S \mid px < qy\}$. (Note that we cannot have $px = qy$.) It is easy to see that $\#S_1 = m$ and $\#S_2 = n$. ■

To demonstrate how we can use the results deduced so far, let us compute $\left(\frac{360}{997}\right)$. Since $360 = 2^3 \cdot 3^2 \cdot 5$, we have

$$\begin{aligned}
\left(\tfrac{360}{997}\right) &= \left(\tfrac{2}{997}\right)\left(\tfrac{5}{997}\right) && \text{(by Proposition 2.21)}\\
&= (-1)^{\frac{997^2-1}{8}}\left(\tfrac{5}{997}\right) = -\left(\tfrac{5}{997}\right) && \text{(by Corollary 2.7)}\\
&= -(-1)^{\frac{5-1}{2}\frac{997-1}{2}}\left(\tfrac{997}{5}\right) = -\left(\tfrac{997}{5}\right) && \text{(by Theorem 2.18)}\\
&= -\left(\tfrac{2}{5}\right) && \text{(by Proposition 2.21)}\\
&= -(-1)^{\frac{5^2-1}{8}} = 1 && \text{(by Corollary 2.7)}\,.
\end{aligned}$$

Thus 360 is a quadratic residue modulo 997. The apparent attractiveness of this method is beset by the fact that it demands the factorization of several integers and as such does not lead to a practical algorithm. We indeed need further machinery in order to have an efficient algorithm. First, we define a generalization of the Legendre symbol.

Definition 2.32 Let a, b be integers with $b > 0$ and odd. We define the *Jacobi symbol* $\left(\frac{a}{b}\right)$ as

$$\left(\frac{a}{b}\right) := \begin{cases} 0 & \text{if } \gcd(a,b) \neq 1,\\ 1 & \text{if } b = 1,\\ \prod_{i=1}^{t}\left(\frac{a}{p_i}\right) & \text{if } \gcd(a,b) = 1 \text{ and } b = p_1 \ldots p_t, \end{cases}$$

where, in the last case, $p_1, \ldots, p_t$ are all the prime factors of b (not necessarily all distinct). □

Note that if $\left(\frac{a}{b}\right) \neq 1$, then a is not a quadratic residue mod b. However, the converse is not always true, that is, $\left(\frac{a}{b}\right) = 1$ does not necessarily imply that a is a quadratic residue modulo b (Example: $a = 2$ and $b = 9$). Of course, if b is an odd prime and if $\gcd(a, b) = 1$, the Legendre and Jacobi symbols $\left(\frac{a}{b}\right)$ correspond to the same value and meaning.

The Jacobi symbol enjoys many properties similar to the Legendre symbol.

Proposition 2.22 For integers a, a' and positive odd integers b, b', we have:

(1) $\left(\frac{aa'}{b}\right) = \left(\frac{a}{b}\right)\left(\frac{a'}{b}\right)$,

(2) $\left(\frac{a}{bb'}\right) = \left(\frac{a}{b}\right)\left(\frac{a}{b'}\right)$, and

(3) if $a \equiv a' \pmod{b}$, then $\left(\frac{a}{b}\right) = \left(\frac{a'}{b}\right)$. In particular, if r is the remainder of Euclidean division of a by b, then $\left(\frac{a}{b}\right) = \left(\frac{r}{b}\right)$.

Proof Immediate from the definition and Proposition 2.21. ■

Theorem 2.19

(1) For an odd positive integer b

$$\left(\frac{-1}{b}\right) = (-1)^{\frac{b-1}{2}} \text{ and } \left(\frac{2}{b}\right) = (-1)^{\frac{b^2-1}{8}}\,.$$

(2) If a is another odd positive integer with $\gcd(a, b) = 1$, then

$$\left(\frac{a}{b}\right)\left(\frac{b}{a}\right) = (-1)^{\frac{a-1}{2}\frac{b-1}{2}}\,.$$

Proof (1) Let $b = p_1 \cdots p_s$, where p_i are odd primes (not necessarily distinct). Then by definition $\left(\frac{-1}{b}\right) = \prod_{i=1}^{s} \left(\frac{-1}{p_i}\right) = (-1)^n$, where $n = \sum_{i=1}^{s} \frac{p_i-1}{2}$. Now for odd integers x and y one has $\frac{x-1}{2} + \frac{y-1}{2} \equiv \frac{xy-1}{2} \pmod 2$. Repeated applications of this prove that $n \equiv \frac{b-1}{2} \pmod 2$. To prove that $\left(\frac{2}{b}\right) = (-1)^{\frac{b^2-1}{8}}$, we proceed in a similar manner and note that for odd integers x and y one has $\frac{x^2-1}{8} + \frac{y^2-1}{8} \equiv \frac{x^2y^2-1}{8} \pmod 2$.

(2) If $a = p'_1 \ldots p'_t$ with $p'_1, \ldots, p'_t$ odd primes, then by definition

$$\left(\frac{a}{b}\right)\left(\frac{b}{a}\right) = \prod_{i=1}^{s}\prod_{j=1}^{t}\left(\frac{p_i}{p'_j}\right)\left(\frac{p'_j}{p_i}\right) = (-1)^m,$$

where from Theorem 2.18 it follows that

$$\begin{aligned} m &\equiv \sum_{i=1}^{s}\sum_{j=1}^{t}\frac{p_i-1}{2}\frac{p'_j-1}{2} = \left(\sum_{i=1}^{s}\frac{p_i-1}{2}\right)\left(\sum_{j=1}^{t}\frac{p'_j-1}{2}\right) \\ &\equiv \frac{b-1}{2}\frac{a-1}{2} \pmod 2. \end{aligned}$$

■

Now, we can calculate $\left(\frac{360}{997}\right)$ without factoring as follows.

$$\begin{aligned} \left(\frac{360}{997}\right) &= \left(\frac{2^3}{997}\right)\left(\frac{45}{997}\right) = \left(\frac{2}{997}\right)\left(\frac{45}{997}\right) = (-1)^{\frac{997^2-1}{8}}\left(\frac{45}{997}\right) \\ &= -\left(\frac{45}{997}\right) = -(-1)^{\frac{45-1}{2}\frac{997-1}{2}}\left(\frac{997}{45}\right) = -\left(\frac{997}{45}\right) = -\left(\frac{7}{45}\right) \\ &= -(-1)^{\frac{7-1}{2}\frac{45-1}{2}}\left(\frac{45}{7}\right) = -\left(\frac{45}{7}\right) = -\left(\frac{3}{7}\right) \\ &= -(-1)^{\frac{3-1}{2}\frac{7-1}{2}}\left(\frac{7}{3}\right) = \left(\frac{7}{3}\right) = \left(\frac{1}{3}\right) = 1. \end{aligned}$$

2.5.4 Some Assorted Topics

So far, we have studied some elementary properties of integers. Number theory is, however, one of the oldest and widest branches of mathematics. Various complex-analytic and algebraic tools have been employed to derive more complicated properties of integers. In Section 2.13, we give a short introductory exposition to *algebraic number theory*. Here, we mention a collection of useful results from *analytic number theory*. The proofs of these analytic results would lead us too far away and hence are omitted here. Inquisitive (and/or cynical) readers may consult textbooks on analytic number theory for the details missing here.

The prime number theorem

The famous *prime number theorem* gives an asymptotic estimate of the density of primes smaller than or equal to a positive real number. Gauss conjectured this result in 1791. Many mathematicians tried to prove it during the 19^{th} century and came up with partial results. Riemann made reasonable progress towards proving the theorem, but could not furnish a complete proof before he died in 1866. It is interesting to mention here that a good portion of the theory of *analytic functions* (also called *holomorphic functions*) in complex analysis was developed during these attempts to prove the prime number theorem. The first complete proof of the theorem (based mostly on the ideas of Riemann and Chebyshev) was given independently by the French mathematician Hadamard and by the Belgian mathematician de la Vallée Poussin in 1896. Their proof is regarded as one of the major achievements of modern mathematics. People started believing that any proof of the prime number theorem has to be analytic. Erdös and Selberg destroyed this belief by independently providing the first *elementary* proof of the theorem in 1949. Here (and elsewhere in mathematics), the adjective *elementary* refers to something which does not depend on results from analysis or algebra. Caution: *Elementary* is not synonymous with *easy*!

Theorem 2.20 **Prime Number Theorem** Let $\pi(x)$ denote the number of primes less than or equal to a real number $x > 0$. As $x \to \infty$ we have $\pi(x) \to x/\ln x$ (that is, the ratio $\pi(x)/(x/\ln x) \to 1$). In particular, for $n \in \mathbb{N}$ the density $\pi(n)/n$ of primes among the natural numbers $\leqslant n$ asymptotically approaches $1/\ln n$ as $n \to \infty$. It also follows that the n-th prime is approximately equal to $n \ln n$. ■

Though the prime number theorem provides an asymptotic estimate (that is, one for $x \to \infty$), for finite values of x (for example, for the values of x in the cryptographic range) it does give good approximations for $\pi(x)$. Table 2.1 lists $\pi(x)$ against the rounded values of $x/\ln x$ for x equal to small powers of 10.

Table 2.1 Approximations to $\pi(x)$

x	$\pi(x)$	$x/\ln x$	$x/(\ln x - 1)$	$\mathrm{Li}(x)$
10^3	168	145	169	178
10^4	1229	1086	1218	1246
10^5	9592	8686	9512	9630
10^6	78,498	72,382	78,030	78,628
10^7	664,579	620,421	661,458	664,918
10^8	5,761,455	5,428,681	5,740,304	5,762,209

Given the prime number theorem it follows that $\pi(x)$ approaches $x/(\ln x - \xi)$ for any real ξ. It turns out that $\xi = 1$ is the best choice. *Gauss'* Li *function* is also an asymptotic estimate for $\pi(x)$, where for real $x > 0$ one defines:

$$\mathrm{Li}(x) := \text{ The principal value of the integral } \int_{\xi=0}^{x} \frac{d\xi}{\ln \xi}$$

Gauss conjectured that $\mathrm{Li}(x)$ asymptotically equals $\pi(x)$. The prime number theorem is, in fact, equivalent to this conjecture. Furthermore, de la Vallée Poussin proved that

$\mathrm{Li}(x)$ is a better approximation to $\pi(x)$ than $x/(\ln x - \xi)$ for any real ξ. Table 2.1 also lists $x/(\ln x - 1)$ and $\mathrm{Li}(x)$ against the actual values of $\pi(x)$.

The asymptotic formula $\lim_{x\to\infty} \frac{\pi(x)}{x/\ln x} = 1$ does not rule out the possibility that the error $\pi(x) - (x/\ln x)$ tends to zero as $x \to \infty$. It has been shown by Dusart [83] that $(x/\ln x) - 0.992(x/\ln^2 x) \leqslant \pi(x) \leqslant (x/\ln x) + 1.2762(x/\ln^2 x)$ for all $x > 598$.

Density of smooth integers

Integers having only small prime divisors play an interesting role in cryptography and in number theory in general.

Definition 2.33 Let $y \in \mathbb{N}$. An integer x is called *y-smooth* (or simply *smooth*, if y is understood from the context), if all the prime divisors of x are $\leqslant y$. We denote by $\psi(x, y)$ the fraction of positive integers $\leqslant x$, that are y-smooth. □

The following theorem gives an asymptotic estimate for $\psi(x, y)$.

Theorem 2.21 Let $x, y \in \mathbb{N}$ with $x > y$ and let $u := \ln x/\ln y$. For $u \to \infty$ and $y \geqslant \ln^2 x$ we have the asymptotic formula:

$$\psi(x, y) \to u^{-u+o(u)} = e^{-[(1+o(1))u \ln u]}.$$ ■

In Theorem 2.21, the notation $g(u) = o(f(u))$ implies that the ratio $g(u)/f(u)$ tends to 0 as u approaches ∞. See Definition 3.1 for more details. An interesting special case of the formula for $\psi(x, y)$ will be used quite often in this book and is given as Corollary 4.1 in Chapter 4.

Like the prime number theorem, Theorem 2.21 gives only asymptotic estimates, but is indeed a good approximation for finite values of x, y and u (that is, for the values of practical interest). The most important implication of this theorem is that the density of y-smooth integers in the set $\{1, \ldots, x\}$ is a very sensitive function of $u = \ln x/\ln y$ and decreases very rapidly as x increases. For example, if $y = 15{,}485{,}863$, the millionth prime, then a random integer $\leqslant 2^{250}$ is y-smooth with probability approximately 2.12×10^{-11}, whereas a random integer $\leqslant 2^{500}$ is y-smooth with probability approximately 2.23×10^{-28}. (These figures are computed neglecting the $o(u)$ term in the expression of $\psi(x, y)$.) In other words, *smaller integers have higher probability of being smooth* (that is, y-smooth for a given y).

The extended Riemann hypothesis

The *Riemann hypothesis* (*RH*) is one of the deepest unsolved problems in mathematics. An extended version of this hypothesis has important bearings on the solvability of certain computational problems in polynomial time.

Definition 2.34 The *Euler zeta function* $\zeta(s)$ is defined for a complex variable s with $\mathrm{Re}\, s \geqslant 1$ as

$$\zeta(s) := \frac{1}{1^s} + \frac{1}{2^s} + \frac{1}{3^s} + \cdots = \sum_{n\in\mathbb{N}} \frac{1}{n^s} = \prod_{p\in\mathbb{P}} \frac{1}{1 - p^{-s}}.$$

The reader may already be familiar with the results: $\zeta(1) = \infty$, $\zeta(2) = \pi^2/6$ and $\zeta(4) = \pi^4/90$. Riemann (analytically) extended the Euler Zeta function for all complex values of s (except at $s = 1$, where the function has a simple pole). This extended function, called the *Riemann zeta function*, is known to have zeros at $s = -2, -4, -6, \ldots$. These are called the *trivial zeros* of $\zeta(s)$. It can be proved that all *non-trivial zeros* of $\zeta(s)$ must lie in the so-called *critical strip*: $0 \leqslant \operatorname{Re} s \leqslant 1$, and are symmetric about the *critical line*: $\operatorname{Re} s = 1/2$. □

Conjecture 2.1 **Riemann hypothesis (RH)** All non-trivial zeros of $\zeta(s)$ lie *on* the critical line. □

In 1900, Hilbert asserted that proving or disproving the RH is one of the most important problems confronting 20^{th} century mathematicians. The problem continues to remain so even to the 21^{st} century mathematicians.

In 1901, von Koch proved that the RH is equivalent to the formula:

Conjecture 2.2 **An equivalent form of the Riemann hypothesis** $\pi(x) = \operatorname{Li}(x) + \mathrm{O}(x^{1/2} \ln x)$ □

Here the order notation $f(x) = \mathrm{O}(g(x))$ means that $|f(x)/g(x)|$ is less than a constant for all sufficiently large x (See Definition 3.1).

Hadamard and de la Vallée Poussin proved that

$$\pi(x) = \operatorname{Li}(x) + \mathrm{O}(xe^{-\alpha\sqrt{\ln x}})$$

for some positive constant α. While this estimate was sufficient to prove the prime number theorem, the tighter bound of Conjecture 2.2 continues to remain unproved.

Theorem 2.22 **Dirichlet's theorem on primes in arithmetic progression** Let $a, b \in \mathbb{N}$ be coprime. The set $\{a + bd \mid d \in \mathbb{N}_0\}$ contains an infinite number of primes. ■

Dirichlet's theorem is a powerful generalization of Theorem 2.12 (which corresponds to $a = b = 1$). One can accordingly generalize the notation $\pi(x)$ as follows:

Definition 2.35 Let $a, b \in \mathbb{N}$ with $\gcd(a, b) = 1$. By $\pi_{a,b}(x)$, we denote the number of primes in the set $\{a + bd \mid d \in \mathbb{N}_0\}$, that are $\leqslant x$. □

The prime number theorem gives the estimate:

$$\pi_{a,b}(x) \to \frac{(1 + \mathrm{o}(1))\, x}{\phi(b) \ln x} \quad \text{as } x \to \infty\,,$$

where ϕ is Euler's totient function. The RH now generalizes to:

Conjecture 2.3 **Extended Riemann hypothesis (ERH)** For $a, b \in \mathbb{N}$ with $\gcd(a, b) = 1$,

$$\pi_{a,b}(x) = \frac{1}{\phi(b)} \operatorname{Li}(x) + \mathrm{O}(x^{1/2} \ln x)\,.$$

□

Some authors use the expression *Generalized Riemann hypothesis* (*GRH*) in place of ERH. Taking $b = 1$ demonstrates that the ERH implies the RH. The ERH also implies the following:

Conjecture 2.4 The smallest positive quadratic non-residue modulo a prime p is $< 2 \ln^2 p$. □

EXERCISE SET 2.5

2.35 **(a)** Show that any integer $n \geqslant 3$ satisfies $n^2 = a^2 - b^2$ for some $a, b \in \mathbb{N}$.
(b) Show that for any integer $n \geqslant 2$ the integer $n^4 + 4^n$ is composite.

2.36 Let $n \in \mathbb{N}$ and S a subset of $\{1, 2, ..., 2n\}$ of cardinality $n + 1$. Show that: **[H]**
(a) There exist $x, y \in S$ such that $x - y = 1$.
(b) There exist $x, y \in S$ such that $x - y = n$.
(c) There exist distinct $x, y \in S$ such that x is a multiple of y.
(d) There exist distinct $x, y \in S$ such that x is relatively prime to y.

2.37 Show that for any $n \in \mathbb{N}$, $n > 1$, the rational number $\sum_{i=1}^{n} \frac{1}{i}$ is not an integer. **[H]**

2.38 **(a)** Show that the *Mersenne number* $M_n := 2^n - 1$ is prime only if n is prime.
(b) Show that the *Fermat number* $2^n + 1$ is prime only if $n = 2^t$ for some $t \in \mathbb{N}$.

2.39 Let $n \geqslant 2$ be a natural number. A *complete residue system* modulo n is a set of n integers $a_1, \ldots, a_n$ such that $a_i \not\equiv a_j \pmod{n}$ for $i \neq j$. Similarly, a *reduced residue system* modulo n is a set of $\phi(n)$ integers $b_1, \ldots, b_{\phi(n)}$ such that $\gcd(b_i, n) = 1$ for all $i = 1, \ldots, \phi(n)$ and $b_i \not\equiv b_j \pmod{n}$ for $i \neq j$. Show that:
(a) If $\{a_1, \ldots, a_n\}$ is a complete residue system modulo n, the equivalence classes of $a_1, \ldots, a_n$ (modulo the ideal $n\mathbb{Z}$) constitute the set $\mathbb{Z}_n$. In other words, given any integer a, there exists a unique i, $1 \leqslant i \leqslant n$, for which $a \equiv a_i \pmod{n}$.
(b) If $\{b_1, \ldots, b_{\phi(n)}\}$ is a reduced residue system modulo n, then the equivalence classes of $b_1, \ldots, b_{\phi(n)}$ constitute the set $\mathbb{Z}_n^*$. In other words, given any integer b coprime to n, there exists a unique i, $1 \leqslant i \leqslant \phi(n)$, for which $b \equiv b_i \pmod{n}$.
(c) If $\{a_1, \ldots, a_n\}$ is a complete residue system modulo n, then for any integer a coprime to n, the integers $aa_1, \ldots, aa_n$ constitute a complete residue system modulo n. For example, if n is odd, then $\{2, 4, 6, \ldots, 2n\}$ is a complete residue system modulo n.
(d) If $\{b_1, \ldots, b_{\phi(n)}\}$ is a reduced residue system modulo n, then for any integer b coprime to n, the integers $bb_1, \ldots, bb_{\phi(n)}$ constitute a reduced residue system modulo n.
(e) For $n > 2$, the integers $1^2, 2^2, \ldots, n^2$ do not constitute a complete residue system modulo n. **[H]**
(f) If p is an odd prime and if $\{a_1, \ldots, a_p\}$ and $\{a'_1, \ldots, a'_p\}$ are two complete residue systems modulo p, then $\{a_1 a'_1, \ldots, a_p a'_p\}$ is not a complete residue system modulo p. **[H]**

2.40 Prove that the decimal expansion of any rational number a/b is recurring, that is, (eventually) periodic. (A terminating expansion may be viewed as one with recurring 0.) **[H]**

2.41 Let p be an odd prime. Show that the congruence $x^2 \equiv -1 \pmod{p}$ is solvable if and only if $p \equiv 1 \pmod{4}$. **[H]**

2.42 Let $n \in \mathbb{N}$.

(a) Show that if $n > 2$, then $\phi(n)$ is even.
(b) Show that if n is odd, then $\phi(n) = \phi(2n)$.
(c) Find out all the values of n for which $\phi(n) = 12$.

2.43 For $n \in \mathbb{N}$, show that $\sum_{d|n, d>0} \phi(d) = n$.

2.44 Let $n > 2$ and $\gcd(a, n) = 1$. Let h be the multiplicative order of a modulo n (that is, in the group $\mathbb{Z}_n^*$). Show that:
(a) $a^i \equiv a^j \pmod{n}$ if and only if $i \equiv j \pmod{h}$.
(b) The multiplicative order of a^l modulo n is $h/\gcd(h, l)$.
(c) If a is a primitive element of $\mathbb{Z}_n^*$ (that is, if $h = \phi(n)$), then $1, a, a^2, \ldots, a^{h-1}$ is a reduced residue system modulo n.
(d) If $\gcd(b, n) = 1$ and b has multiplicative order k modulo n and if $\gcd(h, k) = 1$, then the multiplicative order of ab modulo n is hk.

2.45 Device a criterion for the solvability of $ax^2 + bx + c \equiv 0 \pmod{p}$, where p is an odd prime and $\gcd(a, p) = 1$. **[H]**

2.46 Let p be a prime and $r \in \mathbb{N}$. An integer a with $\gcd(a, p) = 1$ is called an *r-th power residue* modulo p, if the congruence $x^r \equiv a \pmod{p}$ has a solution. Show that a is an r-th power residue modulo p if and only if $a^{(p-1)/\gcd(r,p-1)} \equiv 1 \pmod{p}$. This is a generalization of Euler's criterion for quadratic residues.

2.47 Let G be a finite cyclic group of cardinality n. Show that $G \cong (\mathbb{Z}_n, +)$ and that there are exactly $\phi(n)$ generators (that is, primitive elements) of G.

2.48 Let $m, n \in \mathbb{N}$ with $m|n$. Show that the canonical (surjective) ring homomorphism $\mathbb{Z}_n \to \mathbb{Z}_m$ induces a surjective group homomorphism $\mathbb{Z}_n^* \to \mathbb{Z}_m^*$ of the respective groups of units. (Note that every ring homomorphism $\varphi : A \to B$ induces a group homomorphism $\varphi^* : A^* \to B^*$, where A^* and B^* are the groups of units of A and B respectively. Even when φ is surjective, φ^* need not be surjective, in general. As an example consider the canonical surjection $\mathbb{Z} \to \mathbb{Z}_p = \mathbb{Z}/\langle p \rangle$ for a prime $p > 3$.)

2.49 In this exercise, we investigate which of the groups $\mathbb{Z}_{p^e}$ is cyclic for a prime p and $e \in \mathbb{N}$.
(a) Show that $\mathbb{Z}_2^*$ and $\mathbb{Z}_4^*$ are cyclic, but $\mathbb{Z}_8^*$ is not cyclic. Conclude that $\mathbb{Z}_{2^e}^*$ is not cyclic for $e \geqslant 3$. **[H]** More specifically, show that for $e \geqslant 3$ the multiplicative group $\mathbb{Z}_{2^e}^*$ is the direct product of two cyclic subgroups generated by -1 and 5 respectively.
(b) Show that if p is an odd prime and $e \in \mathbb{N}$, then $\mathbb{Z}_{p^e}^*$ is cyclic. **[H]**

2.50 Show that the multiplicative group $\mathbb{Z}_n^*$, $n \geqslant 2$, is cyclic if and only if $n = 2, 4, p^e, 2p^e$, where p is an odd prime and $e \in \mathbb{N}$. **[H]**

2.6 Polynomials

Unless otherwise stated, in this section we denote by K an arbitrary field and by $K[X]$ the ring of polynomials in one indeterminate X and with coefficients from K. Since

$K[X]$ is a PID, it enjoys many properties similar to those of $\mathbb{Z}$. To start with, we take a look at these properties. Then we introduce the concept of algebraic elements and discuss how irreducible polynomials can be used to construct (algebraic) extensions of fields. When no confusions are likely, we denote a polynomial $f(X) \in K[X]$ by f only.

2.6.1 Elementary Properties

Since $K[X]$ is a PID and hence a UFD, every polynomial in $K[X]$ can be written essentially uniquely as a product of prime polynomials. Conventionally prime polynomials are more commonly referred to as *irreducible polynomials*. Similar to the case of $\mathbb{Z}$ the ring $K[X]$ contains an infinite number of irreducible elements, for if K is infinite, then $\{X - a \mid a \in K\}$ is an infinite set of irreducible polynomials of $K[X]$, and if K is finite, then as we will see later, there is an irreducible polynomial of degree d in $K[X]$ for every $d \in \mathbb{N}$.

It is important to note here that the concept of irreducibility of a polynomial is very much dependent on the field K. If $K \subseteq L$ is a field extension, then a polynomial in $K[X]$ is naturally an element of $L[X]$ also. A polynomial which is irreducible over K need not continue to remain so over L. For example, the polynomial $x^2 - 2$ is irreducible over $\mathbb{Q}$, but reducible over $\mathbb{R}$, since $x^2 - 2 = (x - \sqrt{2})(x + \sqrt{2})$, $\sqrt{2}$ being a real number but not a rational number. As a second example, the polynomial $x^2 + 1$ is irreducible over both $\mathbb{Q}$ and $\mathbb{R}$ but not over $\mathbb{C}$. In fact, we will show shortly that an irreducible polynomial in $K[X]$ of degree > 1 becomes reducible over a suitable extension of K.

For polynomials $f(X), g(X) \in K[X]$ with $g(X) \neq 0$, there exist unique polynomials $q(X)$ and $r(X)$ in $K[X]$ such that $f(X) = q(X)g(X) + r(X)$ with $r(X) = 0$ or $\deg r(X) < \deg g(X)$. The polynomials $q(X)$ and $r(X)$ are respectively called the *quotient* and *remainder* of polynomial division of $f(X)$ by $g(X)$ and can be obtained by the so-called long division procedure. We use the notations: $q(X) = f(X) \operatorname{quot} g(X)$ and $r(X) = f(X) \operatorname{rem} g(X)$.

Whenever we talk about the gcd of two non-zero polynomials, we usually refer to the *monic* gcd, that is, a polynomial with leading coefficient 1. This makes the gcd of two polynomials unique. We have $\gcd(f(X), g(X)) = \gcd(g(X), r(X))$, where $r(X) = f(X) \operatorname{rem} g(X)$. This gives rise to an algorithm (similar to the Euclidean gcd algorithm for integers) for computing the gcd of two polynomials. Bézout relations also hold for polynomials. More specifically:

Proposition 2.23 Let $f(X), g(X) \in K[X]$, not both zero, and $d(X)$ the (monic) gcd of $f(X)$ and $g(X)$. Then there are polynomials $u(X), v(X) \in K[X]$ such that $d(X) = u(X)f(X) + v(X)g(X)$. (Such an equality is called a *Bézout relation*.) Furthermore, if $f(X)$ and $g(X)$ are non-zero and not both constant, then $u(X)$ and $v(X)$ can be so chosen that $\deg u(X) < \deg g(X)$ and $\deg v(X) < \deg f(X)$.[6]

Proof Similar to the proof of Proposition 2.16. ■

The concept of congruence can be extended to polynomials, namely, if $0 \neq f(X) \in K[X]$, then two polynomials $g(X), h(X) \in K[X]$ are said to be *congruent modulo*

[6] Recall that the degree of the zero polynomial is taken to be $-\infty$.

$f(X)$, denoted $g(X) \equiv h(X) \pmod{f(X)}$, if $f(X)|(g(X) - h(X))$, that is, if there exists $u(X) \in K[X]$ with $g(X) - h(X) = u(X)f(X)$, or equivalently, if $g(X) \operatorname{rem} f(X) = h(X) \operatorname{rem} f(X)$.

The principal ideals $\langle f(X)\rangle$ of $K[X]$ play an important role (as do the ideals $\langle n\rangle$ of $\mathbb{Z}$). Let us investigate the structure of the quotient ring $R := K[X]/\langle f(X)\rangle$ for a non-constant polynomial $f(X) \in K[X]$. If $r(X)$ denotes the remainder of division of $g(X) \in K[X]$ by $f(X)$, then it is clear that the residue classes of $g(X)$ and $r(X)$ are the same in R. On the other hand, two polynomials $g(X), h(X) \in K[X]$ with $\deg g(X) < \deg f(X)$ and $\deg h(X) < \deg f(X)$ represent the same residue class in R if and only if $g(X) = h(X)$. Thus elements of R are uniquely representable as polynomials of degrees $< \deg f(X)$. In other words, we may represent the ring R as the set $\{g(X) \in K[X] \mid \deg g(X) < \deg f(X)\}$ together with addition and multiplication modulo the polynomial $f(X)$. The ring R contains all the constant polynomials $a \in K$, that is, the field K is canonically embedded in R. In general, R is not a field. The next theorem gives the criterion for R to be a field.

Theorem 2.23 For a non-constant polynomial $f(X) \in K[X]$, the ring $K[X]/\langle f(X)\rangle$ is a field if and only if $f(X)$ is irreducible in $K[X]$.

Proof If $f(X)$ is reducible over K, then we can write $f(X) = g(X)h(X)$ for some polynomials $g(X), h(X) \in K[X]$ with $1 \leqslant \deg g < \deg f$ and $1 \leqslant \deg h < \deg f$. Then both g and h represent non-zero elements in $K[X]/\langle f(X)\rangle$, whose product is 0, that is, $K[X]/\langle f(X)\rangle$ has non-zero zero divisors.

Conversely, if $f(X)$ is irreducible over K and if $g(X)$ is a non-zero polynomial of degree $< \deg f(X)$, then $\gcd(f(X), g(X)) = 1$, so that by Proposition 2.23 there exist polynomials $u(X), v(X) \in K[X]$ with $u(X)f(X) + v(X)g(X) = 1$ and $\deg v(X) < \deg f(X)$. Thus we see that $v(X)g(X) \equiv 1 \pmod{f(X)}$, that is, $g(X)$ has a multiplicative inverse modulo $f(X)$. ■

Let $L := K[X]/\langle f(X)\rangle$ with $f(X)$ irreducible over K. Then $K \subseteq L$ is a field extension. If $\deg f(X) = 1$, then L is isomorphic to K. If $\deg f(X) \geqslant 2$, then L is a proper extension of K. This gives us a useful and important way of representing the extension field L, given a representation for K. (For example, see Section 2.9.)

2.6.2 Roots of Polynomials

The study of the roots of a polynomial is the central objective in algebra. We now derive some elementary properties of roots of polynomials.

Definition 2.36 Let $f(X) \in K[X]$. An element $a \in K$ is said to be a *root* of f, if $f(a) = 0$. □

Proposition 2.24 Let $f(X) \in K[X]$ and $a \in K$. Then $f(X) = (X - a)q(X) + f(a)$ for some $q(X) \in K[X]$. In particular, a is a root of $f(X)$ if and only if $X - a$ divides $f(X)$.

Proof Polynomial division of $f(X)$ by $X - a$ gives $f(X) = (X - a)q(X) + r(X)$ with $\deg r(X) < \deg(X - a) = 1$. Thus $r(X)$ is a constant polynomial. Let us denote $r(X)$ by $r \in K$. Substituting $X = a$ gives $f(a) = r$. ■

Proposition 2.25 A non-zero polynomial $f \in K[X]$ with $d := \deg f$ can have at most d roots in K.

Proof We proceed by induction on d. The result clearly holds for $d = 0$. So assume that $d \geqslant 1$ and that the result holds for all polynomials of degree $d-1$. If f has no roots in K, we are done. So assume that f has a root, say, $a \in K$. By Proposition 2.24, we have $f(X) = (X-a)g(X)$ for some $g(X) \in K[X]$. Clearly, $\deg g = d-1$ and so by the induction hypothesis g has at most $d-1$ roots. Since K is a field (and hence does not contain non-zero zero divisors), it follows that the roots of f are precisely a and the roots of g. This establishes the induction step. ■

In the last proof, the only result we have used to exploit the fact that K is a field is that K contains no non-zero zero divisors. This is, however, true for every integral domain. Thus Proposition 2.25 continues to hold if K is any integral domain (not necessarily a field). However, if K is not an integral domain, the proposition is not necessarily true. For example, if $ab = 0$ with $a, b \in K \setminus \{0\}$, $a \neq b$, then the polynomial $X^2 + (b-a)X$ has at least three roots: 0, a and $a-b$.

For a field extension $K \subseteq L$ and for a polynomial $f \in K[X]$, we may think of the roots of f in L, since $f \in L[X]$ too. Clearly, all the roots of f in K are also roots of f in L. However, the converse is not true in general. For example, the only roots of $X^4 - 1$ in $\mathbb{R}$ are ± 1, whereas the roots of the same polynomial in $\mathbb{C}$ are $\pm 1, \pm \mathrm{i}$. Indeed we have the following important result.

Proposition 2.26 For any non-constant polynomial $f \in K[X]$, there exists a field extension K' of K such that f has a root in K'.

Proof If f has a root in K, taking $K' = K$ proves the proposition. So we assume that f has no root in K (which implies that $\deg f \geqslant 2$). In principle, we do not require f to be irreducible. But if we consider a non-constant factor g of f, irreducible over K, we see that the roots of g in any extension L of K are roots of f in L too. Thus we may replace f by g and assume, without loss of generality, that f is irreducible. We construct the field extension $K' := K[X]/\langle f \rangle$ of K and denote the equivalence class of X in K' by α. (One also writes x, $\bar{X}$ or $[X]$ to denote this equivalence class.) It is clear that $f(\alpha) = 0 \in K'$, that is, α is a root of $f(X)$ in K'. ■

We say that the field K' in the proof of the last proposition is obtained by *adjoining the root* α of f and denote this as $K' = K(\alpha)$. We can write $f(X) = (X-\alpha)f_1(X)$, where $f_1(X) \in K'[X]$ and $\deg f_1 = (\deg f) - 1$. Now there is a field extension K'' of K', where f_1 has a root. Proceeding in this way we prove the following result.

Proposition 2.27 A non-constant polynomial f in $K[X]$ with $\deg f = d$ has d roots (not necessarily all distinct) in some field extension L of K. ■

If a polynomial $f \in K[X]$ of degree $d \geqslant 1$ has all its roots $\alpha_1, \ldots, \alpha_d$ in L, then $f(X) = a(X-\alpha_1)\cdots(X-\alpha_d)$ for some $a \in L$ (actually $a \in K$). In this case, we say that f *splits* (completely or into linear factors) over L.

Definition 2.37 Let $f \in K[X]$ be a non-constant polynomial. A minimal (with respect to inclusion) field extension of K, over which f splits completely is called a *splitting field* of f over K.[7] This is a minimal field which contains K and all the roots of f. □

[7] It is necessary to use the phrase "over K" in this definition. $X^2 + 1$, treated as a polynomial in $\mathbb{R}[X]$, has the splitting field $\mathbb{C}$, whereas the same polynomial, treated as an element of $\mathbb{Q}[X]$, has the splitting field $\mathbb{Q}(\mathrm{i}) := \{a + \mathrm{i}b \mid a, b \in \mathbb{Q}\}$ (see Equation (2.3) on p 74).

Every non-constant polynomial $f \in K[X]$ has a splitting field L over K. Quite importantly, this field L is unique in some sense. This allows us to call *the* splitting field of f instead of *a* splitting field of f. We discuss these topics further in Section 2.8.

Definition 2.38 Let f be a non-constant polynomial in $K[X]$ and let α be a root of f (in some extension of K). The largest natural number n for which $(X-\alpha)^n | f(X)$ is called the *multiplicity* of the root α (in f). If $n = 1$ (resp. $n > 1$), then α is called a *simple* (resp. *multiple*) *root* of f. If all the roots of f are simple, then we call f a *square-free* polynomial. It is easy to see that f is square-free, only if f is not divisible by the square of a non-constant polynomial in $K[X]$. The reverse implication also holds, if $\operatorname{char} K = 0$ or if K is a finite field (or, more generally, if K is a perfect field—see Exercise 2.76). □

The notion of multiplicity can be extended to a non-root β of f by setting the multiplicity of β to zero.

2.6.3 Algebraic Elements and Extensions

Here we assume, unless otherwise stated, that $K \subseteq L$ is a field extension.

Definition 2.39 An element $\alpha \in L$ is said to be *algebraic over* K, if there exists a non-constant polynomial $f(X) \in K[X]$ with $f(\alpha) = 0$. If an element $\alpha \in L$ is not algebraic over K, we say that α is *transcendental over* K. Thus a transcendental (over K) element $\alpha \in L$ is a root of *no* polynomials in $K[X]$. A field extension $K \subseteq L$ is called an *algebraic extension*, if every element of L is algebraic over K. A non-algebraic extension is also called a *transcendental extension*. If $K \subseteq L$ is a transcendental extension, there exists at least one element $\alpha \in L$, which is transcendental (that is, not algebraic) over K. □

Example 2.10

(1) Every element $\alpha \in K$ is algebraic over K, since it is a root of the non-constant polynomial $X - \alpha \in K[X]$.

(2) The element $\alpha := \sqrt[3]{2+\sqrt{3}} \in \mathbb{R}$ is algebraic over $\mathbb{Q}$, since α is a root of the polynomial $(X^3-2)^2 - 3 = X^6 - 4X^3 + 1 \in \mathbb{Q}[X]$.

(3) The well-known real numbers e and π are transcendental over $\mathbb{Q}$. (We are not going to prove this.) Of course, the concept of algebraic and transcendental elements is heavily dependent on the field K. For example, e and π, being elements of $\mathbb{R}$, are algebraic over $\mathbb{R}$.

(4) A complex number $z = a + \mathrm{i}b \in \mathbb{C}$, where $\mathrm{i} = \sqrt{-1}$ and $a, b \in \mathbb{R}$, is a root of the polynomial $(X-a)^2 + b^2 = X^2 - 2aX + (a^2+b^2) \in \mathbb{R}[X]$ and hence is algebraic over $\mathbb{R}$. Therefore, the field extension $\mathbb{R} \subseteq \mathbb{C}$ is algebraic.

(5) The extension $\mathbb{Q} \subseteq \mathbb{R}$ is transcendental, since $\mathbb{R}$ contains elements (like e and π) that are transcendental over $\mathbb{Q}$. □

Definition 2.40 Let $\alpha \in L$ be algebraic over K. A non-constant polynomial $f \in K[X]$ of least positive degree with $f(\alpha) = 0$ is called a *minimal polynomial* of α over K. □

Proposition 2.28 Let $\alpha \in L$ be algebraic over K. A minimal polynomial f of α over K is irreducible over K. If $h \in K[X]$ is a polynomial with $h(\alpha) = 0$, then $f | h$. In particular, any two minimal polynomials f and g of α satisfy $g(X) = cf(X)$ for some $c \in K^*$.

Proof Let $f = f_1 f_2$ for some non-constant polynomials $f_1, f_2 \in K[X]$. Since K is a field and $0 = f(\alpha) = f_1(\alpha) f_2(\alpha)$, we have $f_1(\alpha) = 0$ or $f_2(\alpha) = 0$. But $\deg f_1 < \deg f$ and $\deg f_2 < \deg f$, a contradiction to the choice of f.

Using polynomial division one can write $h(X) = q(X)f(X) + r(X)$ for some polynomials $q, r \in K[X]$. Now $h(\alpha) = 0$ implies $r(\alpha) = 0$. Since $\deg r < \deg f$, by the choice of f we must then have $r(X) = 0$, that is, $f|h$.

Finally, if f and g are two minimal polynomials of α over K, then $f|g$ and $g|f$ and it follows that $g(X) = cf(X)$ for some unit c of $K[X]$. But the only units of $K[X]$ are the non-zero elements of K. ■

By Proposition 2.28, a *monic* minimal polynomial f of α over K is uniquely determined by α and K. It is, therefore, customary to define *the* minimal polynomial of α over K to be this (unique) monic polynomial. Unless otherwise stated, we will stick to this revised definition and write $f(X) = \mathrm{minpoly}_{\alpha,K}(X)$.

Example 2.11

(1) For $\alpha \in K$, we have $\mathrm{minpoly}_{\alpha,K}(X) = X - \alpha$.

(2) A complex number $z = a + \mathrm{i}b$, $a, b \in \mathbb{R}$, $b \neq 0$, is not a root of a linear polynomial over $\mathbb{R}$, but is a root of the quadratic polynomial $f(X) = X^2 - 2aX + (a^2 + b^2) \in \mathbb{R}[X]$. Therefore, $f = \mathrm{minpoly}_{z,\mathbb{R}}(X)$, that is, f is irreducible over $\mathbb{R}$. □

Proposition 2.29 For a field K, the following conditions are equivalent.

(a) Every proper field extension $K \subsetneq L$ is transcendental (that is, K has no algebraic extensions other than itself).

(b) Every non-constant polynomial in $K[X]$ has a root in K.

(c) Every non-constant polynomial in $K[X]$ splits in K.

(d) Every non-constant irreducible polynomial in $K[X]$ is of degree 1.

Proof [(a)⇒(b)] Consider a non-constant irreducible polynomial $f(X) \in K[X]$ and the field extension $L = K[X]/\langle f \rangle$ of K. We have seen that L contains a root of f. We will prove in Section 2.8 that such an extension is algebraic (Corollary 2.11). Hence (a) implies that $L = K$, that is, K contains a root of f.
[(b)⇒(c)] Let $f \in K[X]$ be a non-constant polynomial. By (b), f has a root, say, $\alpha_1 \in K$. Thus $f(X) = (X - \alpha_1) f_1(X)$ for some $f_1 \in K[X]$ with $\deg f_1 = (\deg f) - 1$. If f_1 is a constant polynomial, we are done. Otherwise, we find as above $\alpha_2 \in K$ and $f_2 \in K[X]$ with $f_1(X) = (X - \alpha_2) f_2(X)$ and with $\deg f_2 = (\deg f) - 2$. Proceeding in this way proves (c).
[(c)⇒(d)] Obvious.
[(d)⇒(a)] Let $\alpha \in L$ be algebraic over K and let $f(X) := \mathrm{minpoly}_{\alpha,K}(X) \in K[X]$. Since f is irreducible, by (d) $\deg f = 1$, that is, $f(X) = X - \alpha$, that is, $\alpha \in K$. ■

Definition 2.41 A field K satisfying the equivalent conditions of Proposition 2.29 is called an *algebraically closed* field. For an arbitrary field K, a minimal algebraically closed field $\bar{K}$ containing K is called an *algebraic closure* of K. □

We will see in Section 2.8 that an algebraic closure of every field exists and is unique in some sense. The algebraic closure of an algebraically closed field K is K itself. We end this section with the following well-known theorem. We will not prove

the theorem in this book, because every known proof of it uses some kind of complex analysis which this book does not deal with.

Theorem 2.24 **Fundamental theorem of algebra** The field $\mathbb{C}$ is algebraically closed. ■

$\mathbb{R}$ is not algebraically closed, since the proper extension $\mathbb{R} \subsetneqq \mathbb{C}$ is algebraic (See Example 2.10). Indeed, $\mathbb{C}$ is the algebraic closure of $\mathbb{R}$.

EXERCISE SET 2.6

2.51 Let R be a ring and $f, g \in R[X]$. Show that:
(a) $\deg(f+g) \leqslant \max(\deg f, \deg g)$ with equality holding, if $\deg f \neq \deg g$.
(b) $\deg(fg) \leqslant \deg f + \deg g$ with equality holding, if R is an integral domain.
(c) If R is an integral domain, then $R[X]$ is an integral domain too. More generally, if R is an integral domain, then $R[X_1, \ldots, X_n]$ is also an integral domain for all $n \in \mathbb{N}$.

2.52 Let $f, g \in R[X]$, where R is an integral domain. Show that if $f(a_i) = g(a_i)$ for $i = 1, \ldots, n$, where $n > \max(\deg f, \deg g)$ and where $a_1, \ldots, a_n$ are *distinct* elements of R, then $f = g$. In particular, if $f(a) = g(a)$ for an infinite number of $a \in R$, then $f = g$.

2.53 **Lagrange's interpolation formula** Let K be a field and let $a_0, \ldots, a_n$ be *distinct* elements of K. Show that for $b_0, \ldots, b_n \in K$ (not necessarily all distinct), there exists a unique polynomial $f(X) \in K[X]$ of degree $\leqslant n$ such that $f(a_i) = b_i$ for all $i = 0, \ldots, n$. **[H]**

2.54 **Polynomials over a UFD** Let R be a UFD. For a non-zero polynomial $f(X) \in R[X]$, a gcd of the coefficients of f is called a *content* of f and is denoted by $\operatorname{cont} f$. One can then write $f = (\operatorname{cont} f) f_1$, where $f_1 \in R[X]$ with $\operatorname{cont} f_1 \in R^*$. f_1 is called a *primitive part* of f and is often denoted as $\operatorname{pp} f$. It is clear that $\operatorname{cont} f$ and $\operatorname{pp} f$ are unique up to multiplication by units of R. If for a non-zero polynomial $f \in R[X]$ the content $\operatorname{cont} f \in R^*$ (or, equivalently, if f and $\operatorname{pp} f$ are associates), then f is called a *primitive polynomial*. Show that for two non-zero polynomials $f, g \in R[X]$ the elements $\operatorname{cont}(fg)$ and $(\operatorname{cont} f)(\operatorname{cont} g)$ are associates in R. In particular, the product of two primitive polynomials is again primitive.

2.55 Let R be a UFD. Show that a non-constant polynomial $f \in R[X]$ is irreducible over R if and only if f is irreducible over $\mathrm{Q}(R)$, where $\mathrm{Q}(R)$ denotes the quotient field of R (see Exercise 2.34).

2.56 **(a) Eisenstein's criterion** Let R be a UFD and $f(X) = a_n X^n + \cdots + a_0 \in R[X]$ with $a_n \neq 0$. Suppose that there is a prime $p \in R$ such that p does not divide a_n, p divides a_i for all i, $0 \leqslant i \leqslant n-1$, and p^2 does not divide a_0. Show that f is irreducible over R.
(b) As an application of Eisenstein's criterion show that for a prime $p \in \mathbb{P}$ the polynomial $X^{p-1} + \cdots + X + 1$ is irreducible in $\mathbb{Z}[X]$. **[H]**

2.57 Let $K \subseteq L$ be a field extension and $f_1, \ldots, f_n$ non-constant polynomials in $K[X]$. Show that each f_i, $i = 1, \ldots, n$, splits over L if and only if the product $f_1 \cdots f_n$ splits over L.

2.58 Show that the irreducible polynomials in $\mathbb{R}[X]$ have degrees $\leqslant 2$. **[H]**

2.59 Show that a finite field (that is, a field with finite cardinality) is not algebraically closed. In particular, the algebraic closure of a finite field is infinite.

2.60 A complex number z is called an *algebraic number*, if z is algebraic over $\mathbb{Q}$. An algebraic number z is called an *algebraic integer*, if z is a root of a *monic* polynomial in $\mathbb{Z}[X]$. Show that:
(a) If z is an algebraic number, then mz is an algebraic integer for some $m \in \mathbb{N}$.
(b) If $a \in \mathbb{Q}$ is an algebraic integer, then $a \in \mathbb{Z}$.
(c) If $z \in \mathbb{C}$ is an algebraic integer, then for any integer $n \in \mathbb{Z}$ the complex numbers nz and $z + n$ are algebraic integers.

2.61 Let K be a field and $f(X) = a_n X^n + \cdots + a_1 X + a_0 \in K[X]$. The *formal derivative* f' of f is defined to be the polynomial $f'(X) := \sum_{j=1}^{n} j a_j X^{j-1} \in K[X]$. Show that:
(a) $(f + g)' = f' + g'$ and $(fg)' = f'g + fg'$ for any $f, g \in K[X]$.
(b) If $\operatorname{char} K = 0$, then $f' = 0$ if and only if $f \in K$.
(c) If $\operatorname{char} K = p > 0$, then $f' = 0$ if and only if $f(X) = g(X^p)$ for some $g(X) \in K[X]$.
(d) f ($\neq 0$) has no multiple roots (in any extension field of K), that is, f is square-free, if and only if $\gcd(f, f') = 1$.
(e) Let f be a (non-constant) irreducible polynomial over K. Show that if $\operatorname{char} K = 0$, then f has no multiple roots. On the other hand, if $\operatorname{char} K = p > 0$, show that f has multiple roots if and only if $f(X) = g(X^p)$ for some $g(X) \in K[X]$. (However, if $K = \mathbb{Z}_p$, then by Fermat's little theorem $g(X^p) = g(X)^p$, which contradicts the fact that $f(x)$ is irreducible. Therefore, f cannot have multiple roots.)

2.62 Let $f(X) \in K[X]$ be a non-constant polynomial of degree d and let $\alpha_1, \ldots, \alpha_d$ be the roots of f (in some extension field of K). The quantity $\Delta(f) := \prod_{1 \leqslant i < j \leqslant d} (\alpha_i - \alpha_j)^2$ is called the *discriminant* of f. Prove the following assertions:
(a) $\Delta(f) = 0$ if and only if f has a multiple root.
(b) $\Delta(f) \in K$.
(c) $\Delta(X^2 + aX + b) = a^2 - 4b$.
(d) $\Delta(X^3 + aX + b) = -(4a^3 + 27b^2)$.

2.7 Vector Spaces and Modules

Vector spaces and linear transformations between them are the central objects of study in linear algebra. In this section, we investigate the basic properties of vector spaces. We also generalize the concept of vector spaces to get another useful class of objects called modules. A module which also carries a (compatible) ring structure is referred to as an algebra. Study of algebras over fields (or more generally over rings) is of importance in commutative algebra, algebraic geometry and algebraic number theory.

2.7.1 Vector Spaces

Unless otherwise specified, K denotes a field in this section.

Definition 2.42 A *vector space V over a field K* (or a *K-vector space*, in short) is an (additively written) Abelian group V together with a multiplication map $\cdot : K \times V \to V$ called the *scalar multiplication* map, such that the following properties are satisfied by every $a, b \in K$ and $x, y \in V$.

(1) $a \cdot (x + y) = a \cdot x + a \cdot y$,
(2) $(a + b) \cdot x = a \cdot x + b \cdot x$,
(3) $1 \cdot x = x$,
(4) $a \cdot (b \cdot x) = (ab) \cdot x$,

where ab denotes the product of a and b in the field K. When no confusions are likely, we omit the scalar multiplication sign $\cdot$ and write $a \cdot x$ simply as ax. □

Example 2.12

(1) Any field K is trivially a K-vector space with the scalar multiplication being the same as the field multiplication. More generally, if $K \subseteq L$ is a field extension, then L is a K-vector space.

(2) For $n \in \mathbb{N}$, the product $K^n = K \times \cdots \times K$ (n factors) is a K-vector space under the scalar multiplication map $a(x_1, \ldots, x_n) := (ax_1, \ldots, ax_n)$. For arbitrary K-vector spaces $V_1, \ldots, V_n$, we can analogously define the product $V_1 \times \cdots \times V_n$.

(3) The polynomial ring $K[X]$ (or $K[X_1, \ldots, X_n]$) is a K-vector space (with the natural scalar multiplication). □

Corollary 2.8 Let V be a K-vector space. For every $a \in K$ and $x \in V$, we have:

(1) $0 \cdot x = 0$.
(2) $a \cdot 0 = 0$.
(3) $(-a) \cdot x = a \cdot (-x) = -(a \cdot x)$.

Proof Easy verification. ■

Definition 2.43 Let V be a vector space over K and S a subset of V. We say that S is a *generating set* or a *set of generators* of V (over K), or that S *generates* V (over K), if every element $x \in V$ can be written as a finite linear combination $x = a_1 x_1 + \cdots + a_n x_n$ for some $n \in \mathbb{N}_0$ (depending on x) and with $a_i \in K$ and $x_i \in S$ for $1 \leqslant i \leqslant n$. A generating set S of V is called *minimal*, if no proper subset of S generates V. If V has a finite generating set, then V is called *finitely generated* or *finite-dimensional*. □

Example 2.13

(1) Consider the field extension $L := K[X]/\langle f(X)\rangle$ of K, where f is an irreducible polynomial in $K[X]$ of degree n. If α denotes the equivalence class of X in L, then every element of L can be written as $a_{n-1}\alpha^{n-1} + \cdots + a_1\alpha + a_0$ with $a_i \in K$ for $0 \leqslant i \leqslant n-1$. Thus $\{1, \alpha, \ldots, \alpha^{n-1}\}$ is a generating set of L over K. In particular, L is finitely generated over K.

(2) The K-vector space K^n is generated by the *unit vectors* e_i, $1 \leqslant i \leqslant n$, defined as $e_i := (0, \ldots, 0, 1, 0, \ldots, 0)$ (1 in the i-th position). Thus K^n is also finitely generated over K.

(3) $\{1, X, X^2, \ldots\}$ is an infinite generating set of the polynomial ring $K[X]$ regarded as a K-vector space. $K[X]$ is *not* finitely generated over K.

It is not difficult to show that the generating sets discussed in these examples are minimal. □

Definition 2.44 A subset S of a K-vector space V is called *linearly independent* (over K), if whenever $a_1x_1 + \cdots + a_nx_n = 0$ for some $n \in \mathbb{N}$, $a_i \in K$ and $x_i \in S$, $1 \leqslant i \leqslant n$, we have $a_1 = \cdots = a_n = 0$. If S is not linearly independent, it is called *linearly dependent*. If S is linearly independent (resp. dependent), then we also say that the elements of S are linearly independent (resp. dependent). A *maximal* linearly independent subset of V is a linearly independent subset $S \subseteq V$ with the property that $S \cup \{x\}$ is linearly dependent for any $x \in V \setminus S$. □

If $0 \in S$, then S is linearly dependent, since $a \cdot 0 = 0$ for any $a \in K$. One can easily check that all the generating sets of Example 2.13 are linearly independent too. This is, however, not a mere coincidence, as the following result demonstrates.

Theorem 2.25 A subset S of a K-vector space V is a minimal generating set for V if and only if S is a maximal linearly independent set of V.

Proof [if] Given a maximal linearly independent subset S of V, we first show that S is a generating set for V. Take any non-zero $x \in V \setminus S$. By the maximality of S, the set $S \cup \{x\}$ is linearly dependent, that is, there exists a linear relation of the form $a_0x + a_1x_1 + \cdots + a_nx_n = 0$, $a_i \in K$, $x_i \in S$, with some $a_i \neq 0$. The linear independence of S forces $a_0 \neq 0$ and so $x = -a_0^{-1}a_1x_1 - \cdots - a_0^{-1}a_nx_n$ is a finite linear combination of elements of S. Thus S generates V. Now, we show that S is minimal. Assume otherwise, that is, $S' := S \setminus \{y\}$ generates V for some $y \in S$. Since S is linearly independent, $y \neq 0$. For some $m \in \mathbb{N}$, $b_i \in K$, $y_i \in S'$, we then have $y = b_1y_1 + \cdots + b_my_m$, a contradiction to the linear independence of S.
[only if] Given a minimal generating set S of V, we first show that S is linearly independent. Assume not, that is, $a_1x_1 + \cdots + a_nx_n = 0$ for some $n \in \mathbb{N}$, $a_i \in K$ and $x_i \in S$ with some a_i, say a_1, non-zero. But then $x_1 = -a_1^{-1}a_2x_2 - \cdots - a_1^{-1}a_nx_n$ and, therefore, $S \setminus \{x_1\}$ also generates V, a contradiction to the minimality of S. Thus S is linearly independent. Now choose a non-zero $y \in V \setminus S$. Since S generates V, we can write $y = b_1y_1 + \cdots + b_my_m$, $m \in \mathbb{N}$, $b_i \in K$ and $y_i \in S$, that is, $1y - b_1y_1 - \cdots - b_my_m = 0$, that is, $S \cup \{y\}$ is linearly dependent. ■

Definition 2.45 Let V be a K-vector space. A minimal generating set S of V is called a *basis* of V over K (or a K-*basis* of V). By Theorem 2.25, S is a basis of V if and only if S is a maximal linearly independent subset of V. Equivalently, S is a basis of V if and only if S is a generating set of V and is linearly independent. □

Any element of a vector space can be written *uniquely* as a finite linear combination of elements of a basis, since two different ways of writing the same element contradict the linear independence of the basis elements.

A K-vector space V may have many K-bases. For example, the elements $1, aX + b, (aX + b)^2, \ldots$ form a K-basis of $K[X]$ for any $a, b \in K$, $a \neq 0$. However, what is unique in any basis of a given K-vector space V is the cardinality[8] of the basis, as shown in Theorem 2.26.

For the sake of simplicity, we sometimes assume that V is a finitely generated K-vector space. This assumption simplifies certain proofs greatly. But it is important to highlight here that, unless otherwise stated, all the results continue to remain valid without the assumption. For example, it is a fact that every vector space has a basis. For finitely generated vector spaces, this is a trivial statement to prove, whereas without our assumption we need to use arguments that are not so simple. (A possible proof follows from Exercise 2.63 with $U = \{0\}$.)

Theorem 2.26 Let V be a K-vector space. Then any K-basis of V has the same cardinality.

Proof We assume that V is finitely generated. Let $S = \{x_1, \ldots, x_n\}$ be a minimal finite generating set, that is, a basis, of V. Let T be another basis of V. Assume that $m := \#T > n$. (We might even have $m = \infty$.) We can choose distinct elements $y_1, \ldots, y_{n+1} \in T$. Note that x_i and y_j are non-zero. Now we can write $y_1 = a_1x_1 + \cdots + a_nx_n$ for some (unique) $a_i \in K$, with some $a_i \neq 0$. Renumbering $x_1, \ldots, x_n$, if necessary, we may assume that $a_1 \neq 0$. Then $x_1 = a_1^{-1}y_1 - a_1^{-1}a_2x_2 - \cdots - a_1^{-1}a_nx_n$. It follows that $y_1, x_2, \ldots, x_n$ generate V. In particular, we can write $y_2 = b_1y_1 + b_2x_2 + \cdots + b_nx_n$, $b_i \in K$, with some $b_i \neq 0$. If $b_2 = \cdots = b_n = 0$, then y_1, y_2 are linearly dependent, a contradiction. So $b_i \neq 0$ for some i, $2 \leqslant i \leqslant n$. Again we may renumber $x_2, \ldots, x_n$, if necessary, to assume that $b_2 \neq 0$. Then $x_2 = b_2^{-1}y_2 - b_2^{-1}b_1y_1 - b_2^{-1}b_3x_3 - \cdots - b_2^{-1}b_nx_n$, that is, $y_1, y_2, x_3, \ldots, x_n$ generate V. Proceeding in this way we can show that $y_1, \ldots, y_n$ generate V, a contradiction to the minimality of T as a generating set. Thus we must have $m \leqslant n$. In particular, m is finite. Now reversing the roles of S and T we can likewise prove that $n \leqslant m$. ■

Theorem 2.26 holds even when V is not finitely generated. We omit the proof for this case here.

Definition 2.46 Let V be a K-vector space. The cardinality of any K-basis of V is called the *dimension* of V over K and is denoted by $\dim_K V$ (or by $\dim V$, if K is understood from the context). We call V *finite-dimensional* (resp. *infinite-dimensional*), if $\dim_K V$ is finite (resp. infinite). □

For example, $\dim_K K^n = n$, $\dim_K(V_1 \times \cdots \times V_n) = \sum_{i=1}^n \dim_K V_i$, and $\dim_K K[X] = \infty$.

Definition 2.47 Let V be a K-vector space. A subgroup U of V, which is closed under the scalar multiplication of V, is again a K-vector space and is called a (*vector*) *subspace* of V. In this case, we have $\dim_K U \leqslant \dim_K V$ (Exercise 2.63). □

Example 2.14 Let V be a vector space over K.

(1) The subset $\{0\}$ and V are trivially subspaces of V.

[8] Two sets (finite or not) S_1 and S_2 are said to be of the same cardinality, if there exists a bijective map $S_1 \to S_2$.

(2) Let S be any subset of V (not necessarily linearly independent). Then the set $U := \{\sum_{i=1}^{n} a_i x_i \mid n \in \mathbb{N}_0, a_i \in K, x_i \in S\}$ is a vector subspace of V. We say that U is *spanned* or *generated* by S, or that S *generates* or *spans* U, or that U is the *span* of S. This is often denoted by $U = \sum_{x \in S} Kx$ or by $U = \operatorname{Span} S$. If S is linearly independent, then S is a basis of U. □

Definition 2.48 Let V and W be K-vector spaces. A map $f : V \to W$ is called a *homomorphism* (of vector spaces) or a *linear transformation* or a *linear map* over K, if

$$f(ax + by) = af(x) + bf(y)$$

for all $a, b \in K$ and $x, y \in V$. Equivalently, f is a linear map over K if and only if $f(x + y) = f(x) + f(y)$ and $f(ax) = af(x)$ for all $a \in K$ and $x, y \in V$. The set of all K-linear maps $V \to W$ is denoted by $\operatorname{Hom}_K(V, W)$. $\operatorname{Hom}_K(V, W)$ is a K-vector space under the definitions $(f + g)(x) := f(x) + g(x)$ and $(af)(x) := af(x)$ for all $f, g \in \operatorname{Hom}_K(V, W)$ and $x \in V$. A K-linear transformation $V \to V$ is called a K-*endomorphism* of V. The set of all K-endomorphisms of V is denoted by $\operatorname{End}_K V$. A bijective[9] homomorphism (resp. endomorphism) is called an *isomorphism* (resp. *automorphism*). □

Theorem 2.27 Let V and W be K-vector spaces. Then V and W are isomorphic if and only if $\dim_K V = \dim_K W$.

Proof If $\dim_K V = \dim_K W$ and S and T are bases of V and W respectively, then there exists a bijection $f : S \to T$. One can extend f to a linear map $\tilde{f} : V \to W$ as $\tilde{f}(\sum_{i=1}^{n} a_i x_i) := \sum_{i=1}^{n} a_i f(x_i)$, for $n \in \mathbb{N}_0$, $a_i \in K$ and $x_i \in S$. One can readily verify that $\tilde{f}$ is an isomorphism. Conversely, if $g : V \to W$ is an isomorphism and S is any basis of V, then $g(S) := \{g(x) \mid x \in S\}$ is clearly a basis of W. ■

Corollary 2.9 A K-vector space V with $n := \dim_K V < \infty$ is isomorphic to K^n. ■

Let V be a K-vector space and U a subspace. As in Section 2.3 we construct the quotient group V/U. This group can be given a K-vector space structure under the scalar multiplication map $a(x + U) := ax + U$, $a \in K$, $x \in V$. If $T \subseteq V$ is such that the residue classes of the elements of T form a K-basis of V/U and if S is a K-basis of U, then it is easy to see that $S \cup T$ is a K-basis of V. In particular,

$$\dim_K V = \dim_K U + \dim_K (V/U)\,. \tag{2.2}$$

For $f \in \operatorname{Hom}_K(V, W)$, the set $\{x \in V \mid f(x) = 0\}$ is called the *kernel* $\operatorname{Ker} f$ of f, and the set $\{y \in W \mid y = f(x) \text{ for some } x \in V\}$ is called the *image* $\operatorname{Im} f$ of f. We have the isomorphism theorem for vector spaces:

Theorem 2.28 **Isomorphism theorem** $\operatorname{Ker} f$ is a subspace of V, $\operatorname{Im} f$ is a subspace of W, and $V/\operatorname{Ker} f \cong \operatorname{Im} f$.

Proof Similar to Theorem 2.3 and Theorem 2.9. ■

[9] As in Footnote 2, we continue to be lucky here: The inverse of a bijective linear transformation is again a linear transformation.

Definition 2.49 For $f \in \mathrm{Hom}_K(V,W)$, the dimension of $\mathrm{Im}\, f$ is called the *rank* of f and is denoted by $\mathrm{Rank}\, f$, whereas the dimension of $\mathrm{Ker}\, f$ is called the *nullity* of f and is denoted by $\mathrm{Null}\, f$. An immediate consequence of the isomorphism theorem and of Equation (2.2) is the following important result. □

Theorem 2.29 $\mathrm{Rank}\, f + \mathrm{Null}\, f = \dim_K V$ for any $f \in \mathrm{Hom}_K(V,W)$. ■

* 2.7.2 Modules

If we remove the restriction that K is a field and assume that K is any ring, then a *vector space* over K is called a K-module. More specifically, we have:

Definition 2.50 Let R be a ring. A *module over* R (or an R*-module*) is an (additively written) Abelian group M together with a multiplication map $\cdot : R \times M \to M$ called the *scalar multiplication* map, such that for every $a, b \in R$ and $x, y \in M$ we have $a \cdot (x+y) = a \cdot x + a \cdot y$, $(a+b) \cdot x = a \cdot x + b \cdot x$, $1 \cdot x = x$, and $a \cdot (b \cdot x) = (ab) \cdot x$, where ab denotes the product of a and b in the ring R. When no confusions are likely, we omit the scalar multiplication sign $\cdot$ and write $a \cdot x$ as ax. □

Example 2.15

(1) Vector spaces are special cases of modules, when the underlying ring is a field.

(2) Ideals of R are modules over R with the ring multiplication map taken as the scalar multiplication.

(3) Every Abelian group G is a $\mathbb{Z}$-module under the scalar multiplication

$$n \cdot x := \begin{cases} 0 & \text{if } n = 0\,, \\ x + \cdots + x \ (n \text{ times}) & \text{if } n > 0\,, \\ -((-n)x) & \text{if } n < 0\,. \end{cases}$$

(4) The polynomial rings $R[X]$ and $R[X_1, \ldots, X_n]$ are modules over R.

(5) Let M_i, $i \in I$, be a family of R-modules. The *direct product* $\prod_{i\in I} M_i$ of M_i is defined as the set of all tuples indexed by I. The *direct sum* $\bigoplus_{i\in I} M_i$ is the subset of the Cartesian product $\prod_{i\in I} M_i$ consisting only of the tuples $(a_i)_{i\in I}$ for which $a_i = 0$ except for a *finite* number of $i \in I$. Both the direct product and the direct sum are R-modules under component-wise addition and scalar multiplication. When I is finite, they are naturally the same. □

Modules are a powerful generalization of vector spaces. Any result we prove for modules is equally valid for vector spaces, ideals and Abelian groups. On the other hand, since we do not demand that the ring R be necessarily a field, certain results for vector spaces are not applicable for all modules.

It is easy to see that Corollary 2.8 continues to hold for modules. An R*-submodule* of an R-module M is a subgroup of M, that is closed under the scalar multiplication of M. For a subset $S \subseteq M$, the set of all finite linear combinations of the form $a_1x_1 + \cdots + a_nx_n$, $n \in \mathbb{N}_0$, $a_i \in R$, $x_i \in S$, is an R-submodule N of M, denoted by RS or $\sum_{x\in S} Rx$. We say that N is *generated by* S (or by the elements of S). If S is finite,

then N is said to be *finitely generated*. A (sub)module generated by a single element is called *cyclic*. It is important to note that unlike vector spaces the cardinality of a minimal generating set of a module is not necessarily unique. (See Exercise 2.68 for an example.) It is also true that given a minimal generating set S of M, there may be more than one ways of writing an element of M as finite linear combinations of elements of S. For example, if $M = R = \mathbb{Z}$ and $S = \{2, 3\}$, then $1 = (-1)\cdot 2 + 1\cdot 3 = 2\cdot 2 + (-1)\cdot 3$. The nice theory of dimensions developed in connection with vector spaces does not apply to modules.

For an R-submodule N of M, the Abelian group M/N is given an R-module structure by the scalar multiplication map $a(x + N) := ax + N$. This module is called the *quotient module* of M by N.

For R-modules M and N, an *R-linear map* or an *R-module homomorphism* (from M to N) is defined as a map $f : M \to N$ with $f(ax+by) = af(x)+bf(y)$ for all $a, b \in R$ and $x, y \in M$ (or equivalently with $f(x+y) = f(x) + f(y)$ and $f(ax) = af(x)$ for all $a \in R$ and $x, y \in M$). An *isomorphism*, an *endomorphism* and an *automorphism* are defined in analogous ways as in case of vector spaces. The set of all (R-module) homomorphisms $M \to N$ is denoted by $\mathrm{Hom}_R(M, N)$ and the set of all (R-module) endomorphisms of M is denoted by $\mathrm{End}_R M$. These sets are again R-modules under the definitions: $(f + g)(x) := f(x) + g(x)$ and $(af)(x) := af(x)$ for all $a \in R$ and $x \in M$ (and f, g in $\mathrm{Hom}_R(M, N)$ or $\mathrm{End}_R M$).

The *kernel* and *image* of an R-linear map $f : M \to N$ are defined as the sets $\mathrm{Ker}\, f := \{x \in M \mid f(x) = 0\}$ and $\mathrm{Im}\, f := \{y \in N \mid y = f(x)$ for some $x \in M\}$. With these notations we have the isomorphism theorem for modules:

Theorem 2.30 **Isomorphism theorem** $\mathrm{Ker}\, f$ and $\mathrm{Im}\, f$ are submodules of M and N respectively and $M/\mathrm{Ker}\, f \cong \mathrm{Im}\, f$. ■

For an R-module M and an ideal $\mathfrak{a}$ of R, the set $\mathfrak{a}M$ consisting of all finite linear combinations $\sum_{i=1}^{n} a_i x_i$ with $n \in \mathbb{N}_0$, $a_i \in \mathfrak{a}$ and $x_i \in M$ is a submodule of M. On the other hand, for a submodule N of M the set $(M : N) := \{a \in R \mid aM \subseteq N\}$ is an ideal of R. In particular, the ideal $(M : 0)$ is called the *annihilator* of M and is denoted as $\mathrm{Ann}_R M$ (or as $\mathrm{Ann}\, M$). For any ideal $\mathfrak{a} \subseteq \mathrm{Ann}\, M$, one can view M as an $R/\mathfrak{a}$-module under the map $(a+\mathfrak{a})x := ax$. One can easily check that this map is well-defined, that is, the product $(a + \mathfrak{a})x$ is independent of the choice of the representative a of the equivalence class $a + \mathfrak{a}$.

Definition 2.51 A *free module* M over a ring R is defined to be a direct sum $\bigoplus_{i\in I} M_i$ of R-modules M_i with each $M_i \cong R$ as an R-module. If I is of finite cardinality n, then M is isomorphic to R^n. □

Any vector space is a free module (Theorem 2.27 and Corollary 2.9). The Abelian groups ($\mathbb{Z}$-modules) $\mathbb{Z}_n$, $n \in \mathbb{N}$, are not free.

Theorem 2.31 **Structure theorem for finitely generated modules** M is a finitely generated R-module if and only if M is a quotient of a free module R^n for some $n \in \mathbb{N}_0$.

Proof [if] The free module R^n has a canonical generating set e_i, $i \in I$, where

$$e_i = (0, \ldots, 0, 1, 0, \ldots, 0) \quad (1 \text{ in the } i\text{-th position}).$$

If $M = R^n/N$, then the equivalence classes $e_i + N$, $i = 1, \ldots, n$, constitute a finite set of generators of M.

[only if] If $x_1, \ldots, x_n$ generate M, then the R-linear map $f : R^n \to M$ defined by $(a_1, \ldots, a_n) \mapsto a_1x_1 + \cdots + a_nx_n$ is surjective. Hence by the isomorphism theorem $M \cong R^n/\operatorname{Ker} f$. ■

** 2.7.3 Algebras

Let $\varphi : R \to A$ be a homomorphism of rings. The ring A can be given an R-module structure with the multiplication map $a \cdot x := \varphi(a)x$ for $a \in R$ and $x \in A$. This R-module structure of A is compatible with the ring structure of A in the sense that for every $a, b \in R$ and $x, y \in A$ one has $(ax)(by) = (ab)(xy)$.

Conversely, if a ring A has an R-module structure with $(ax)(by) = (ab)(xy)$ for every $a, b \in R$ and $x, y \in A$, then there is a unique ring homomorphism $\varphi : R \to A$ taking $a \mapsto a \cdot 1$ (where 1 denotes the identity of A). This motivates us to define the following.

Definition 2.52 Let R be a ring. An *algebra over* R or an *R-algebra* is a ring A together with a ring homomorphism $\varphi : R \to A$. The homomorphism φ is called the *structure homomorphism* of the R-algebra A. If A and B are R-algebras with structure homomorphisms $\varphi : R \to A$ and $\psi : R \to B$, then an *R-algebra homomorphism* (from A to B) is a ring homomorphism $\eta : A \to B$ such that $\psi = \eta \circ \varphi$. □

Example 2.16 Let R be a ring.

(1) The polynomial ring $R[X_1, \ldots, X_n]$ is an R-algebra with the canonical inclusion as the structure homomorphism and is called a *polynomial algebra* over R.

(2) For an ideal $\mathfrak{a}$ of R, the canonical surjection $R \to R/\mathfrak{a}$ makes $R/\mathfrak{a}$ an R-algebra.

(3) If A is an R-algebra with structure homomorphism $\varphi : R \to A$ and if B is an A-algebra with structure homomorphism $\psi : A \to B$, then B is an R-algebra with structure homomorphism $\psi \circ \varphi$.

(4) Combining (2) and (3) implies that if A is an R-algebra and $\mathfrak{a}$ an ideal of A, then the ring $A/\mathfrak{a}$ is again an R-algebra, called the *quotient algebra* of A by $\mathfrak{a}$. □

An R-algebra A is an R-module with the added property that multiplication of elements of A is now legal. Exploiting this new feature leads to the following concept of algebra generators.

Definition 2.53 Let A be an R-algebra with the structure homomorphism $\varphi : R \to A$. A subset S of A is said to generate A as an R-algebra, if every element $x \in A$ can be written as a polynomial expression in (finitely many) elements of S with coefficients from R (that is, from $\varphi(R)$). We write this as $A = R[S]$. If $S = \{x_1, \ldots, x_n\}$ is finite, we also write $R[x_1, \ldots, x_n]$ in place of $R[S]$ and say that A is *finitely generated* as an R-algebra or that the homomorphism $\varphi : R \to A$ is *of finite type*. □

Example 2.17

(1) The polynomial algebra $R[X_1, \ldots, X_n]$, $n \geqslant 1$, over R is not finitely generated as an R-module, but is finitely generated as an R-algebra.

(2) For an ideal $\mathfrak{a}$ of $R[X_1, \ldots, X_n]$, the ring $A := R[X_1, \ldots, X_n]/\mathfrak{a}$ is generated as an R-algebra by the equivalence classes $x_i := X_i + \mathfrak{a}$ of X_i, $1 \leqslant i \leqslant n$, that is, $A = R[x_1, \ldots, x_n]$. If $\mathfrak{a}$ is not the zero ideal, then A is not a polynomial algebra, because $x_1, \ldots, x_n$ are not indeterminates in the sense that they satisfy (non-zero) polynomial equations $f(x_1, \ldots, x_n) = 0$ for every $f \in \mathfrak{a}$. (In this case, we also say that $x_1, \ldots, x_n$ are *algebraically dependent*.) The notation $R[\ldots]$ is a generalization of the notation for polynomial algebras. In what follows, we usually denote polynomial algebras by $R[X_1, \ldots, X_n]$ with upper-case algebra generators, whereas for an arbitrary finitely generated R-algebra we use lower-case symbols for the algebra generators as in $R[x_1, \ldots, x_n]$. □

One may proceed to define kernels and images of R-algebra homomorphisms and frame and prove the isomorphism theorem for R-algebras. We leave the details to the reader. We only note that algebra homomorphisms are essentially ring homomorphisms with the added condition of commutativity with the structure homomorphisms.

Theorem 2.32 A ring A is a finitely generated R-algebra if and only if A is a quotient of a polynomial algebra (over R).

Proof [if] Immediate from Example 2.17.
[only if] Let $A := R[x_1, \ldots, x_n]$. The map $\eta : R[X_1, \ldots, X_n] \to A$ that takes $f(X_1, \ldots, X_n) \mapsto f(x_1, \ldots, x_n)$ is a surjective R-algebra homomorphism. By the isomorphism theorem, one has the isomorphism $A \cong R[X_1, \ldots, X_n]/\operatorname{Ker}\eta$ of R-algebras. ■

This theorem suggests that for the study of finitely generated algebras it suffices to investigate only the polynomial algebras and their quotients.

EXERCISE SET 2.7

2.63 Let V be a K-vector space, U a subspace of V, and T an arbitrary K-basis of U. Show that there is a K-basis of V, that contains T. **[H]**

2.64 **(a)** Let V be a K-vector space, and U_1, U_2 subspaces of V. Show that the set $U := \{x + y \mid x \in U_1, y \in U_2\}$ is a K-subspace of V. If $U_1 \cap U_2 = \{0\}$, we say that U is the *direct sum* of U_1 and U_2 and write $U = U_1 \oplus U_2$.
(b) Let V be a K-vector space and W a subspace of V. Show that there exists a subspace W' of V such that $V = W \oplus W'$. This space W' is called the *complement subspace* of W in V. **[H]**

2.65 Let V and W be K-vector spaces and $f : V \to W$ a K-linear map. Show that f is uniquely determined by the images $f(x)$, $x \in S$, where S is a basis of V.

2.66 Let V and W be K-vector spaces. Check that $\operatorname{Hom}_K(V, W)$ is a vector space over K. Show that $\dim_K(\operatorname{Hom}_K(V, W)) = (\dim_K V)(\dim_K W)$. In particular, if $W = K$, then $\operatorname{Hom}_K(V, K)$ is isomorphic to V. The space $\operatorname{Hom}_K(V, K)$ is called the *dual space* of V.

2.67 Let V and W be m- and n-dimensional K-vector spaces, $S = \{x_1, \ldots, x_m\}$ a K-basis of V, $T = \{y_1, \ldots, y_n\}$ a K-basis of W, and $f : V \to W$ a K-linear map. For

each $i = 1, \ldots, m$, write $f(x_i) = a_{i1}y_1 + \cdots + a_{in}y_n$, $a_{ij} \in K$. The $m \times n$ matrix $\mathcal{T}_f := (a_{ij})$ is called the *transformation matrix* of f (with respect to the bases S and T). We have:

$$f\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{pmatrix} = \begin{pmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_m) \end{pmatrix} = \mathcal{T}_f \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}.$$

Let V_1, V_2, V_3 be K-vector spaces, $f, f_1, f_2 \in \mathrm{Hom}_K(V_1, V_2)$, $g \in \mathrm{Hom}_K(V_2, V_3)$ and $\alpha \in K$. Prove the following assertions:
(a) $\mathcal{T}_{f_1+f_2} = \mathcal{T}_{f_1} + \mathcal{T}_{f_2}$.
(b) $\mathcal{T}_{\alpha f} = \alpha \mathcal{T}_f$.
(c) $\mathcal{T}_{g \circ f} = \mathcal{T}_f \times \mathcal{T}_g$.
(d) f is invertible (as a map) if and only if $\mathcal{T}_f$ is invertible (as a matrix).
(**Remark:** This exercise explains that the linear transformations of finite-dimensional vector spaces can be explained in terms of matrices.)

2.68 Show that for every $n \in \mathbb{N}$ there are integers $a_1, \ldots, a_n$ that constitute a *minimal* set of generators for the unit ideal in $\mathbb{Z}$. **[H]**

2.69 Let M be an R-module. A subset S of M is called a *basis* of M, if S generates M and is linearly independent over R in the sense that $\sum_{i=1}^{n} a_i x_i = 0$, $n \in \mathbb{N}$, $a_i \in R$, $x_i \in S$, implies $a_1 = \cdots = a_n = 0$. Show that M has a basis if and only if M is a free R-module.

2.70 We define the *rank* of a finitely generated R-module M as

$$\mathrm{Rank}_R M := \min\{\#S \mid M \text{ is generated by } S\}.$$

If N is a submodule of M, show that $\mathrm{Rank}_R M \leqslant \mathrm{Rank}_R N + \mathrm{Rank}_R(M/N)$. Give an example where the strict inequality holds.

2.71 Let M be an R-module. An element $x \in M$ is called a *torsion element* of M, if $\mathrm{Ann}\, Rx \neq 0$, that is, if there is $0 \neq a \in R$ with $ax = 0$. The set of all torsion elements of M is denoted by $\mathrm{Tors}\, M$. M is called *torsion-free* if $\mathrm{Tors}\, M = \{0\}$, and a *torsion module* if $\mathrm{Tors}\, M = M$.
(a) Show that $\mathrm{Tors}\, M$ is a submodule of M.
(b) Show that $\mathrm{Tors}\, M$ is a torsion module (called the *torsion submodule* of M) and that the module $M/\,\mathrm{Tors}\, M$ is torsion-free.
(c) If R is an integral domain, show that every free module over R is torsion-free. In particular, every vector space is torsion-free.

2.72 Show that:
(a) $\mathbb{Q}$ is not finitely generated as a $\mathbb{Z}$-module. **[H]**
(b) $\mathbb{Q}$ is not a free $\mathbb{Z}$-module. **[H]**
(c) $\mathbb{Q}$ is a torsion-free $\mathbb{Z}$-module.
This shows that the converse of Exercise 2.71(c) is not true in general.

2.8 Fields

In this section, we study some important properties of field extensions. We also give an introduction to Galois theory. Unless otherwise stated, the letters F, K and L stand for fields in this section.

2.8.1 Properties of Field Extensions

We have seen that if $F \subseteq K$ is a field extension, then K is a vector space over F. This observation leads to the following very useful definitions.

Definition 2.54 For a field extension $F \subseteq K$, the cardinality of any F-basis of K is called the *degree of the extension* $F \subseteq K$ and is denoted by $[K : F]$. If $[K : F]$ is finite, K is called a *finite extension* of F. Otherwise, K is called an *infinite extension* of F. □

Proposition 2.30 Let $F \subseteq K \subseteq L$ be a tower of field extensions. Then $[L : F] = [L : K]\,[K : F]$. In particular, the extension $F \subseteq L$ is finite if and only if the extensions $F \subseteq K$ and $K \subseteq L$ are finite. In that case, $[L : K] \,|\, [L : F]$ and $[K : F] \,|\, [L : F]$.

Proof One can easily check that if S is an F-basis of K and S' a K-basis of L, then the set $T := \{xy \mid x \in S \text{ and } y \in S'\}$ is an F-basis of L. ■

Recall the definitions of the rings $F[X]$ of polynomials and $F(X)$ of rational functions in one indeterminate X. These notations are now generalized. For a field extension $F \subseteq K$ and for $a \in K$, we define:

$$F[a] := \{f(a) \mid f(X) \in F[X]\}$$

and

$$F(a) := \{f(a)/g(a) \mid f(X), g(X) \in F[X], g(a) \neq 0\}\,. \tag{2.3}$$

It is easy to see that $F[a]$ is the smallest (with respect to inclusion) of the integral domains that contain F and a. Similarly $F(a)$ is the smallest of the fields that contain F and a. We also have $F[a] \subseteq F(a)$. Now we state the following important characterization of algebraic elements.

Theorem 2.33 For a field extension $F \subseteq K$ and an element $a \in K$, the following conditions are equivalent:

(a) The element a is algebraic over F.
(b) The extension $F(a)$ is finite over F.
(c) $F(a) = F[a]$.

Proof [(a)⇒(b)] Let $h(X) := \mathrm{minpoly}_{a,F}(X) \in F[X]$ be of degree d. Consider the ring homomorphism $\varphi : F[X] \to F(a)$ that takes $f(X) \mapsto f(a)$. From Proposition 2.28, $\mathrm{Ker}\,\varphi = \langle h \rangle$, and by the isomorphism theorem $F[X]/\langle h \rangle \cong \mathrm{Im}\,\varphi$. Since h is irreducible over F, $F[X]/\langle h \rangle$ and so $\mathrm{Im}\,\varphi$ are fields. Since $\mathrm{Im}\,\varphi$ contains F and a (note

that $\varphi(X) = a$), we have $F(a) \subseteq \operatorname{Im}\varphi$, that is, $\operatorname{Im}\varphi = F(a)$. Finally, notice that $[F[X]/\langle h\rangle : F] = d$.
[(b)⇒(c)] Let $d := [F(a) : F]$. Since the elements $1, a, a^2, \ldots, a^d$ are linearly dependent over F, there exists $\alpha_0, \ldots, \alpha_d \in F$, not all 0, such that $\alpha_0 + \alpha_1 a + \cdots + \alpha_d a^d = 0$. This, in turn, implies that there is an irreducible polynomial $h(X) \in F[X]$ with $h(a) = 0$. Now consider $f(a)/g(a) \in F(a)$. Clearly, $h \not| g$ (because otherwise $g(a) = 0$). Since h is irreducible, $\gcd(g, h) = 1$, that is, there exist polynomials $u(X), v(X) \in F[X]$ with $u(X)g(X) + v(X)h(X) = 1$, that is, with $u(a)g(a) = 1$. But then $f(a)/g(a) = f(a)u(a) \in F[a]$.
[(c)⇒(a)] Clearly, the element 0 is algebraic over F. So assume $a \neq 0$. Since $1/a \in F(a)$, by hypothesis there is a polynomial $f(X) \in F[X]$ such that $1/a = f(a)$. But then a is a root of the non-constant polynomial $Xf(X) - 1 \in F[X]$. ■

Corollary 2.10 For a field extension $F \subseteq K$, the set of elements in K that are algebraic over F is a field.

Proof It is sufficient to show that if $a, b \in K$ are algebraic over F, then the elements $a \pm b$, ab and a/b (if $b \neq 0$) are also algebraic over F. By Theorem 2.33, $[F(a) : F]$ is finite. Since b is algebraic over F, it is also algebraic over $F(a)$. In particular, $[F(a)(b) : F(a)]$ is finite. But then the extension $F(a)(b)$ is also finite over F and contains $a \pm b$, ab and a/b (if $b \neq 0$). ■

The field $F(a)(b)$ in the proof of the last corollary is also denoted as $F(a, b)$. It is the smallest subfield of K that contains F, a and b, and it follows that $F(a, b) = F(b, a)$. More generally, for a field extension $F \subseteq K$ and for $a_1, \ldots, a_n \in K$, each algebraic over F, the field $F(a_1, \ldots, a_n)$ is defined as $F(a_1)(a_2)\cdots(a_n)$ and is independent of the order in which a_i are adjoined.

Corollary 2.11 Let $F \subseteq K$ be a finite extension. Then K is algebraic over F.

Proof For any $a \in K$, we have $F \subseteq F(a) \subseteq K$. Now use Proposition 2.30. ■

The converse of the last corollary is not true, that is, it is possible that an algebraic extension has infinite extension degree. Exercise 2.59 gives an example.

Corollary 2.12 If $F \subseteq K$ and $K \subseteq L$ are algebraic field extensions, then $F \subseteq L$ is also algebraic.

Proof Take an arbitrary $a \in L$. Since $K \subseteq L$ is algebraic, there is a non-zero polynomial $f(X) := \alpha_n X^n + \alpha_{n-1}X^{n-1} + \cdots + \alpha_0 \in K[X]$ such that $f(a) = 0$. It then follows that a is algebraic over $F(\alpha_0, \ldots, \alpha_n)$. Since each α_i is algebraic over F, the degree $[F(\alpha_0, \ldots, \alpha_n) : F]$ is finite. Therefore, $[F(\alpha_0, \ldots, \alpha_n)(a) : F] = [F(\alpha_0, \ldots, \alpha_n)(a) : F(\alpha_0, \ldots, \alpha_n)]\,[F(\alpha_0, \ldots, \alpha_n) : F]$ is also finite and hence $F(\alpha_0, \ldots, \alpha_n)(a)$ and, in particular, a are algebraic over F. ■

Definition 2.55 A field extension $F \subseteq K$ is called *simple*, if $K = F(a)$ for some $a \in K$. □

Proposition 2.31 Let F be a field of characteristic 0 and let a, b (belonging to some extension of F) be algebraic over F. Then the extension $F(a, b)$ of F is simple.

Proof Let $p(X)$ and $q(X)$ be the minimal polynomials (over F) of a and b respectively. Let $d := \deg p$ and $d' := \deg q$. The polynomials p and q are irreducible over F and

hence by Exercise 2.61 have no multiple roots. Let $a_1, \ldots, a_d$ be the roots of p and $b_1, \ldots, b_{d'}$ the roots of q with $a = a_1$ and $b = b_1$. For each i, j with $j \neq 1$, the equation $a_i + \lambda b_j = a + \lambda b$ has a unique solution for λ (not necessarily in F). Since F is infinite, we can choose $\mu \in F$ which is not a solution of any of the equations just mentioned. Define $c := a + \mu b$, so that $c \neq a_i + \mu b_j$ for all i, j with $j \neq 1$. Clearly, $F(c) \subseteq F(a, b)$. To prove the reverse inclusion, note that by hypothesis $q(b) = 0$. Also if we define $f(X) := p(c - \mu X) \in F(c)[X]$, we see that $f(b) = p(a) = 0$. By the choice of c, we have $f(b_j) \neq 0$ for $j \neq 1$. Finally since q is square-free, we have $\gcd(q, f) = X - b \in F(c)[X]$. This implies that $b \in F(c)$ and so $a = c - \mu b \in F(c)$ too. ■

Corollary 2.13 A finite extension $F \subseteq K$ of fields of characteristic 0 is simple.

Proof We proceed by induction on $d := [K : F]$. The result vacuously holds for $d = 1$. So let us assume that $d > 1$ and that the result holds for all smaller values of d. Choose an element $a \in K \setminus F$. Then $[F(a) : F] > 1$ and divides d. If $[F(a) : F] = d$, we are done. So assume $[F(a) : F] < d$. Since $[K : F(a)] < d$, by the induction hypothesis the extension $F(a) \subseteq K$ is simple, say $K = F(a)(b) = F(a, b)$. The result now follows immediately from the previous proposition. ■

2.8.2 Splitting Fields and Algebraic Closure

Let $f(X)$ be a non-constant polynomial of degree d in $F[X]$. Assume that f does nor split over F. Consider an irreducible (in $F[X]$) factor f' of f of degree $d' > 1$. $F' := F[X]/\langle f' \rangle$ is a field extension of F. Furthermore, if $\alpha_1 := X + \langle f'(X) \rangle \in F'$, the elements $1, \alpha_1, \ldots, \alpha_1^{d'-1}$ constitute a basis of F' over F. In particular, $[F' : F] = d' \leqslant d$. Now, one can write $f(X) = (X - \alpha_1)g(X)$ for some $g \in F'[X]$. If g splits over F', so does f too. Otherwise, choose any irreducible (in $F'[X]$) factor g' of g with $\deg g' > 1$ and consider the field extension $F'' := F'[X]/\langle g' \rangle$. Then $[F'' : F'] = \deg g' \leqslant \deg g = d - 1$, so that $[F'' : F] \leqslant d(d - 1)$. Moreover, if $\alpha_2 := X + \langle g'(X) \rangle \in F''$, then $f(X) = (X - \alpha_1)(X - \alpha_2)h(X)$ for some $h \in F''[X]$. Proceeding in this way we get:

Proposition 2.32 For a polynomial $f \in F[X]$ of degree $d \geqslant 1$, there is a field extension K of F with $[K : F] \leqslant d!$, such that f splits over K. ■

We now establish the *uniqueness* of the splitting field of a polynomial $f \in F[X]$. To start with, we set up certain notations. An isomorphism $\mu : F \to F'$ of fields induces an isomorphism $\mu^* : F[X] \to F'[Y]$ of polynomial rings, defined by $a_d X^d + a_{d-1} X^{d-1} + \cdots + a_0 \mapsto \mu(a_d) Y^d + \mu(a_{d-1}) Y^{d-1} + \cdots + \mu(a_0)$. We have $\mu^*(a) = \mu(a)$ for all $a \in F$. Note also that $f \in F[X]$ is irreducible over F if and only if $\mu^*(f) \in F'[Y]$ is irreducible over F'. With these notations we state the following important lemma.

Lemma 2.5 Let the non-constant polynomial $f \in F[X]$ be irreducible over F. Let α and β be roots of f and $\mu^*(f)$ respectively. Then there is an isomorphism $\tau : F(\alpha) \to F'(\beta)$ of fields such that $\tau(a) = \mu(a)$ for all $a \in F$ and $\tau(\alpha) = \beta$.

Proof Since $F(\alpha) = F[\alpha]$ and $F'(\beta) = F'[\beta]$, we can define the map $\tau : F[\alpha] \to F'[\beta]$ by $g(\alpha) \mapsto (\mu^*(g))(\beta)$ for each $g \in F[X]$. It is now an easy check that τ is a well-defined isomorphism of fields with the desired properties. ■

Roots of an irreducible polynomial are called *conjugates* (of each other). If α and β are two roots of an irreducible polynomial $f(X) \in F[X]$, the last lemma guarantees the existence of an isomorphism $\tau : F(\alpha) \to F(\beta)$ that fixes all the elements of F and that maps $\alpha \mapsto \beta$.

Proposition 2.33 We use the maps $\mu : F \to F'$ and $\mu^* : F[X] \to F'[Y]$ as defined above. Let $f(X) \in F[X]$ be a non-constant polynomial and let K and K' be splitting fields of f and $\mu^*(f)$ over F and F' respectively. Then there is an isomorphism $\tau : K \to K'$ of fields, such that $\tau(a) = \mu(a)$ for all $a \in F$.

Proof We proceed by induction on $n := [K : F]$. (By Proposition 2.32 n is finite.) If $n = 1$, then $K = F$, that is, the polynomial f splits over F itself and so does $\mu^*(f)$ over F', that is, $K' = F'$. Thus $\tau = \mu$ is the desired isomorphism.

Now assume that $n > 1$ and that the result holds for all fields L and for all polynomials in $L[X]$ with splitting fields (over L) of extension degrees less than n. Consider an irreducible factor g of f with $1 < \deg g \leqslant \deg f$. Note that g also splits over K. We take any root $\alpha \in K$ of g and consider the tower of field extensions $F \subseteq F(\alpha) \subseteq K$. Similarly, let $\beta \in K'$ be a root of $\mu^*(g)$ and consider $F' \subseteq F'(\beta) \subseteq K'$. By Lemma 2.5 there is an isomorphism $\nu : F(\alpha) \to F'(\beta)$ with $\nu(a) = \mu(a)$ for all $a \in F$ and $\nu(\alpha) = \beta$. Now $[K : F(\alpha)] = [K : F]/[F(\alpha) : F] = [K : F]/\deg g < n$. It is evident that K and K' are splitting fields of f and $\mu^*(f)$ over $F(\alpha)$ and $F'(\beta)$ respectively. Hence by the induction hypothesis there is an isomorphism $\tau : K \to K'$ with $\tau(a) = \nu(a)$ for all $a \in F(\alpha)$. In particular, $\tau(a) = \mu(a)$ for all $a \in F$. ■

The results pertaining to the splitting field of a polynomial can be generalized in the following way. Let S be a non-empty subset of $F[X]$. A splitting field of S over F is a minimal field K containing F such that each polynomial $f \in S$ splits in K. If $S = \{f_1, \ldots, f_r\}$ is a finite set, the splitting field of S is the same as the splitting field of $f = f_1 \cdots f_r$ (Exercise 2.57). But the situation is different, if S is infinite. Of particular interest is the set S consisting of *all* irreducible polynomials in $F[X]$. In this case, the splitting field of S is an algebraic closure of F.

We give a sketch of the proof that even when S is infinite, a splitting field for S can be constructed. This, in particular, establishes the existence of an algebraic closure of any field. We may assume that S comprises non-constant polynomials only. For each $f \in S$, we define an indeterminate X_f and consider the ring $A := F[X_f \mid f \in S]$ and the ideal $\mathfrak{a}$ of A generated by $f(X_f)$ for all $f \in S$. We have $\mathfrak{a} \neq A$ and, therefore, there is a maximal ideal $\mathfrak{m}$ of A containing $\mathfrak{a}$ (Exercise 2.23). Consider the field $F_1 := A/\mathfrak{m}$ containing F. Every polynomial $f \in S$ contains at least one root in F_1. Now we replace F by F_1 and as above get another field F_2 containing F_1 (and hence F), such that every polynomial in S (of degree $\geqslant 2$) has at least two roots in F_2. We continue this procedure (infinitely often, if necessary) and obtain a sequence of fields $F \subseteq F_1 \subseteq F_2 \subseteq F_3 \subseteq \cdots$. Define K to be the field consisting of all elements of $\bigcup_{n\in\mathbb{N}} F_n$, that are algebraic over F. Each polynomial in S splits in K, but in no proper subfield of K, that is, K is a splitting field of S.

It turns out that the splitting field of S is unique up to isomorphisms that fix elements of F. In particular, *the* algebraic closure of F is unique up to isomorphisms that fix elements of F, and is denoted by $\bar{F}$.

* 2.8.3 Elements of Galois Theory

For a field K, the set $\mathrm{Aut}\, K$ of all automorphisms of K is a group under (functional) composition. We extend this concept now. Let $F \subseteq K$ be an extension of fields.

Definition 2.56 An automorphism $\varphi \in \mathrm{Aut}\, K$ is called an *F-automorphism of K*, if φ fixes all the elements of F (which means that $\varphi(a) = a$ for all $a \in F$). The set of all F-automorphisms of K is denoted by $\mathrm{Aut}_F\, K$ or by $\mathrm{Gal}(K|F)$ and is a subgroup of $\mathrm{Aut}\, K$. The *Galois group* of a polynomial $f \in F[X]$ is defined to be the group $\mathrm{Aut}_F\, K$, where K is the splitting field of f over F.

Conversely, for a subgroup H of $\mathrm{Aut}_F\, K$ the set of elements of K that are fixed by all the automorphisms of H, that is, the set of all $a \in K$ with $\varphi(a) = a$ for every $\varphi \in H$, is a subfield of K, called the *fixed field* of H (over F) and denoted as $\mathrm{Fix}_F\, H$. Clearly, $F \subseteq \mathrm{Fix}_F\, H \subseteq K$. □

For every intermediate field L (that is, a field L with $F \subseteq L \subseteq K$), we have a subgroup $\mathrm{Aut}_L\, K$ of $\mathrm{Aut}_F\, K$. Conversely, given a subgroup H of $\mathrm{Aut}_F\, K$ we have the intermediate fixed field $\mathrm{Fix}_F\, H$. It is a relevant question to ask if there is any relationship between the subgroups of $\mathrm{Aut}_F\, K$ and the intermediate fields. A nice correspondence exists for a particular type of extensions that we define now.

Definition 2.57 A field extension $F \subseteq K$ is said to be a *Galois extension* (or K is said to be a Galois extension over F), if $\mathrm{Fix}_F(\mathrm{Aut}_F\, K) = F$. Thus K is Galois over F if and only if for every $\alpha \in K \setminus F$ there is a $\varphi \in \mathrm{Aut}_F\, K$ with $\varphi(\alpha) \neq \alpha$. □

Example 2.18 Let K be the splitting field of a non-constant polynomial $f \in F[X]$. By Exercise 2.77, the extension $F \subseteq K$ is normal. Assume that $F \subseteq K$ is a separable extension (Exercise 2.75). Consider an element $\alpha \in K \setminus F$ and let g be the minimal polynomial of α over F. Then $\deg g > 1$ and g splits in $K[X]$. By assumption (of separability), there is a root $\beta \in K$ of g with $\beta \neq \alpha$. Lemma 2.5 shows that there is a $\tau \in \mathrm{Aut}_F\, K$ such that $\tau(\alpha) = \beta$. Thus, K is Galois over F. In particular, if $\mathrm{char}\, F = 0$ or if $F = \mathbb{Z}_p$, then $F \subseteq K$ is separable and so Galois. For example, $\mathbb{C}$ is a Galois extension of $\mathbb{R}$. □

The following theorem establishes the correspondence we are looking for.

Theorem 2.34 **Fundamental theorem of Galois theory** For a finite Galois extension $F \subseteq K$, there is a bijective correspondence between the set of all intermediate fields and the set of all subgroups of $\mathrm{Aut}_F\, K$ (given by $L \mapsto \mathrm{Aut}_L\, K$ and $H \mapsto \mathrm{Fix}_F\, H$) such that the following assertions hold:

(1) $\mathrm{Aut}_{\mathrm{Fix}_F\, H}\, K = H$ for every subgroup H of $\mathrm{Aut}_F\, K$.
(2) $\mathrm{Fix}_F(\mathrm{Aut}_L\, K) = L$ for every field L with $F \subseteq L \subseteq K$.
(3) For field extensions $F \subseteq L \subseteq L' \subseteq K$, the extension degree $[L' : L]$ is the same as the index $[\mathrm{Aut}_L\, K : \mathrm{Aut}_{L'}\, K]$. In particular, the order of $\mathrm{Aut}_F\, K$ is $[K : F]$.
(4) For every intermediate field L, one has:

(a) K is Galois over L.

(b) L is Galois over F if and only if $\mathrm{Aut}_L\, K$ is a normal subgroup of $\mathrm{Aut}_F\, K$. In this case, $\mathrm{Aut}_F\, L \cong \mathrm{Aut}_F\, K / \mathrm{Aut}_L\, K$. ■

A proof of this theorem is rather long and uses many auxiliary results which we would not need otherwise. We, therefore, choose to omit the proof here.

EXERCISE SET 2.8

2.73 Let α be transcendental over F. Show that the domain $F[\alpha]$ and the field $F(\alpha)$ are respectively isomorphic to the polynomial ring $F[X]$ and the field $F(X)$ of rational functions in one *indeterminate* X. Generalize the result for an arbitrary family α_i, $i \in I$, of elements each of which is transcendental over F.

2.74 Let $F \subseteq K$ be a field extension and let φ be an endomorphism of K with $\varphi(a) = a$ for every $a \in F$.
(a) If a non-constant polynomial $f(X) \in F[X]$ has a root $\alpha \in K$, show that $\varphi(\alpha) \in K$ is also a root of f. For example, if $F = \mathbb{R}$, $K = \mathbb{C}$ and φ is the automorphism mapping z to its (complex) conjugate $\bar{z}$, then we conclude that if a complex number z is a root of $f(X) \in \mathbb{R}[X]$, then $\bar{z}$ is also a root of f. A similar result holds for the extension $\mathbb{Q} \subseteq \mathbb{Q}(\sqrt{m})$, where m is a non-square rational number.
(b) If K is algebraic over F, show that φ is an automorphism. **[H]**

2.75 Let $F \subseteq K$ be a field extension.
(a) An irreducible polynomial $f \in F[X]$ is said to be *separable* over F, if f has no multiple roots. An algebraic element $\alpha \in K$ is said to be *separable* over F, if the minimal polynomial of α over F is separable. K is called a *separable extension* of F, if every element of K is (algebraic and) separable over F. Show that if $\mathrm{char}\, F = 0$ or if $F = \mathbb{Z}_p$, and if K is an algebraic extension of F, then K is separable over F. **[H]**
(b) An algebraic element $\alpha \in K$ is called *purely inseparable* over F, if the minimal polynomial of α over F factors in $K[X]$ as $(X - \alpha)^n$ for some $n \in \mathbb{N}$. If every element of K is (algebraic and) purely inseparable over F, then K is called a *purely inseparable extension* of F. Show that $\alpha \in K$ is both separable and purely inseparable if and only if $\alpha \in F$. Thus, if $\mathrm{char}\, F = 0$ or $F = \mathbb{Z}_p$, then F has no purely inseparable extension other than itself.
(c) If $p := \mathrm{char}\, F > 0$, then an element $\alpha \in K$ is purely inseparable over K if and only if $\mathrm{minpoly}_{\alpha,F}(X) = X^{p^r} + a$ for some $r \geqslant 0$ and $a \in F$. In particular, show that if K is a finite purely inseparable extension of F, then $[K : F] = p^s$ for some $s \geqslant 0$.

2.76 F is called a *perfect field*, if every irreducible polynomial in $F[X]$ is separable over F.
(a) Show that F is a perfect field if and only if every algebraic extension of F is separable over F. In particular, the fields of characteristic 0 and the fields $\mathbb{F}_p$, $p \in \mathbb{P}$, are perfect.
(b) Let $p := \mathrm{char}\, F > 0$. Show that F is perfect if and only if every element of F has a p-th root in F. **[H]**

2.77 A field extension $F \subseteq K$ is called *normal*, if every irreducible polynomial in $F[X]$, that has a root in K, splits in $K[X]$.
(a) If K is the splitting field of a polynomial $f(X) \in F[X]$ over F, show that K is a normal extension of F. **[H]**
(b) If $[K : F] = 2$, show that $F \subseteq K$ is a normal extension.
(c) Consider the tower $\mathbb{Q} \subseteq \mathbb{Q}(\sqrt{2}) \subseteq \mathbb{Q}(\sqrt[4]{2})$ of field extensions to conclude that if $F \subseteq K$ and $K \subseteq L$ are normal extensions, then $F \subseteq L$ need not be normal.

2.78 Prove the following assertions:
(a) $\mathbb{R}$ is an infinite extension of $\mathbb{Q}$. **[H]**
(b) $\mathrm{Aut}_{\mathbb{Q}} \mathbb{R} = \{\mathrm{id}_{\mathbb{R}}\}$. **[H]**

2.79 Let $F \subseteq K$ be a field extension and let L be the fixed field of $\mathrm{Aut}_F K$ over F. Show that K is a Galois extension of L.

2.9 Finite Fields

Finite fields are seemingly the most important types of fields used in cryptography. They enjoy certain nice properties that infinite fields (in particular, the well-known fields like $\mathbb{Q}$, $\mathbb{R}$ and $\mathbb{C}$) do not. We concentrate on some properties of finite fields in this section. As we see later, arithmetic over a finite field K is *fast*, when $\mathrm{char}\, K = 2$ or when $\#K$ is a prime. As a result, these two classes of fields are the most common ones employed in cryptography. However, in this section, we do not restrict ourselves to these specific fields only, but provide a general treatment valid for all finite fields. As in the previous section, we continue to use the letters F, K, L to denote fields. In addition, we use the letter p to denote a prime number and q a power of p: that is, $q = p^n$ for some $n \in \mathbb{N}$.

2.9.1 Existence and Uniqueness of Finite Fields

Let K be a finite field of cardinality q. Then $p := \mathrm{char}\, K > 0$. By Proposition 2.7, p is a prime, that is, K contains an isomorphic copy of the field $F := \mathbb{Z}_p$. If $n := [K : F] \in \mathbb{N}$, we have $q = p^n$. Therefore, we have proved the first statement of the following important result.

Theorem 2.35 The cardinality of a finite field is a power p^n, $n \in \mathbb{N}$, of a prime number p. Conversely, given $p \in \mathbb{P}$ and $n \in \mathbb{N}$, there exists a finite field of cardinality p^n.

Proof In order to construct a finite field of cardinality $q := p^n$, we start with $F := \mathbb{Z}_p$ and consider *the* splitting field K of the polynomial $f(X) := X^q - X \in F[X]$. Since $f'(X) = -1 \neq 0$, the roots of f are distinct (Exercise 2.61). Therefore, the set $E := \{a \in K \mid a^q = a\}$ has cardinality q. By Exercise 2.80, E is a field. Since $F \subseteq E \subseteq K$ and f splits over E, by definition of splitting fields, we have $K = E$, that is, $\#K = \#E = q$. ■

Theorem 2.36 **Fermat's little theorem for finite fields** Let K be a finite field of cardinality q. Then every $a \in K$ satisfies $a^q = a$.

Proof Clearly, $0^q = 0$. Take $a \neq 0$. K^* being a group of order $q-1$, by Proposition 2.4 $\mathrm{ord}_{K^*}(a)$ divides $q-1$. In particular, $a^{q-1} = 1$, that is, $a^q = a$. ■

Theorem 2.37 Let K be a finite field of cardinality $q = p^n$ and let F be the subfield of K isomorphic to $\mathbb{Z}_p$. Then K is the splitting field of the polynomial $f(X) := X^q - X \in F[X]$ over F. In particular, K is unique up to F-isomorphisms (that is, isomorphisms fixing elements of F).

Proof By Theorem 2.37, each of the q elements of K is a root of f and consequently K is the splitting field of f. The last assertion in the theorem follows from the uniqueness of splitting fields (Proposition 2.33). ■

This uniqueness allows us to talk about *the* finite field of cardinality q (rather than *a* finite field of cardinality q). We denote this (unique) field by $\mathbb{F}_q$.

The results proved so far can be generalized for arbitrary extensions $\mathbb{F}_q \subseteq \mathbb{F}_{q^m}$, where $q = p^n$, $n, m \in \mathbb{N}$. We leave the details to the reader (Exercise 2.82). It is important to point out here that since $\mathbb{F}_{q^m}$ is the splitting field of $X^{q^m} - X$ over $\mathbb{F}_q$, by Exercise 2.77 we have:

Corollary 2.14 Every finite extension of finite fields is normal. ■

This implies that an irreducible polynomial $f(X) \in \mathbb{F}_q[X]$ has either none or all of its roots in $\mathbb{F}_{q^m}$. Also if $\alpha \in \mathbb{F}_q$ with $q = p^n$, then $\alpha^q = \alpha^{p^n} = \alpha$. Therefore, $\alpha^{p^{n-1}}$ is a p-th root of α. By Exercise 2.76(b), we then conclude:

Corollary 2.15 Every finite field is perfect. ■

Proposition 2.34 Consider the extension $\mathbb{F}_q \subseteq \mathbb{F}_{q^m}$, $m \in \mathbb{N}$. There is a unique intermediate field with q^d elements, $d \in \mathbb{N}$, if and only if $d|m$. Furthermore, if $d|m$, then $\alpha \in \mathbb{F}_{q^m}$ belongs to the (unique intermediate) field $\mathbb{F}_{q^d}$ if and only if $\alpha^{q^d} = \alpha$.

Proof For $d|m$, we have $(X^{q^d} - X)|(X^{q^m} - X)$. The q^d roots of $X^{q^d} - X$ in K constitute an intermediate field L. If $L' \neq L$ is another intermediate field with q^d elements, by Theorem 2.36 there are more than q^d elements of K, that are roots of $X^{q^d} - X$, a contradiction. Conversely, an intermediate field L contains q^d elements, where $d := [L : \mathbb{F}_q]$. Since $m = [\mathbb{F}_{q^m} : \mathbb{F}_q] = [\mathbb{F}_{q^m} : L][L : \mathbb{F}_q]$, we have $d|m$. The last assertion in the proposition follows immediately from the above argument. ■

Corollary 2.16 Let $\alpha \in \mathbb{F}_{q^m}$ and $f(X) := \mathrm{minpoly}_{\alpha,\mathbb{F}_q}(X) \in \mathbb{F}_q[X]$. Then $\deg f$ divides m.

Proof Consider the extension $\mathbb{F}_q[X]/\langle f \rangle \cong \mathbb{F}_{q^d}$ of $\mathbb{F}_q$, where $d := \deg f$, and the fact that $\mathbb{F}_q \subseteq \mathbb{F}_{q^m}$ is a normal extension. ■

Now we will prove a very important result concerning the multiplicative group $\mathbb{F}_q^*$.

Theorem 2.38 $\mathbb{F}_q^*$ is a cyclic group for any finite field $\mathbb{F}_q$.

Proof Modify the proof of Proposition 2.19 or use the following more general result. ■

Theorem 2.39 Let K be a field (not necessarily finite). Then any *finite* subgroup G of the multiplicative group K^* is cyclic.

Proof Since K is a field, for any $n \in \mathbb{N}$ the polynomial $X^n - 1$ has at most n roots in K and hence in G. The theorem then follows immediately from Exercise 2.18. ■

Corollary 2.17 Every finite extension $\mathbb{F}_q \subseteq \mathbb{F}_{q^m}$ is simple. In particular, $\mathbb{F}_q[X]$ contains an irreducible polynomial of degree m (for any q and m).

Proof Let α be a generator of the cyclic group $\mathbb{F}^*_{q^m}$. Then, m is the smallest of the positive integers s for which $\alpha^{q^s} = \alpha$. Let $f(X) := \text{minpoly}_{\alpha,\mathbb{F}_q}(X)$ with $d := \deg f$, so that $\mathbb{F}_q \subseteq \mathbb{F}_q(\alpha) = \mathbb{F}_q[X]/\langle f\rangle = \mathbb{F}_{q^d} \subseteq \mathbb{F}_{q^m}$. If $d < m$, then $\alpha^{q^d} = \alpha$, a contradiction. Thus $d = m$, that is, $\mathbb{F}_{q^m} = F_q(\alpha)$. ■

2.9.2 Polynomials over Finite Fields

In this section, we study some useful properties of polynomials over finite fields. We concentrate on polynomials in $\mathbb{F}_q[X]$ for an arbitrary $q = p^n$, $p \in \mathbb{P}$, $n \in \mathbb{N}$. We have seen how the polynomials $X^{q^m} - X$ proved to be important for understanding the structures of finite fields. But that is not all; these polynomials indeed have further roles to play. This prompts us to reserve the following special symbol: $\mathcal{I}_{q,m}(X) := X^{q^m} - X \in \mathbb{F}_q[X]$.

Let $\mathbb{F}_q \subseteq \mathbb{F}_{q^m}$ be a finite extension of finite fields and let $\alpha \in \mathbb{F}_{q^m}$ be a root of the polynomial $f(X) = a_d X^d + a_{d-1}X^{d-1} + \cdots + a_0 \in \mathbb{F}_q[X]$. Since each $a_i \in \mathbb{F}_q$, we have $a_i^q = a_i$. Therefore, $f(\alpha^q) = a_d\alpha^{qd} + a_{d-1}\alpha^{q(d-1)} + \cdots + \alpha_0 = a_d^q\alpha^{qd} + a_{d-1}^q\alpha^{q(d-1)} + \cdots + \alpha_0^q = f(\alpha)^q = 0$. More generally, for each $r = 0, 1, 2, \ldots$ the element $\alpha^{q^r} \in \mathbb{F}_{q^m}$ is a root of $f(X)$. This gives us a nice procedure for computing the minimal polynomial of α as the following corollary suggests.

Corollary 2.18 The minimal polynomial of $\alpha \in \mathbb{F}_{q^m}$ over $\mathbb{F}_q$ is $(X - \alpha)(X - \alpha^q)\cdots(X - \alpha^{q^{d-1}})$, where d is the smallest of the integers $s \in \mathbb{N}$ for which $\alpha^{q^s} = \alpha$.

Proof Let $f_\alpha(X) := \text{minpoly}_{\alpha,\mathbb{F}_q}(X)$ have degree δ. So $\mathbb{F}_{q^\delta}$ is the smallest field containing ($\mathbb{F}_q$ and) α and hence all the roots of f_α, that is, $\alpha^{q^s} = \alpha$ for $s = \delta$ and for no smaller positive integer values of s. Therefore, $\delta = d$ and all the conjugates of α are precisely $\alpha, \alpha^q, \ldots, \alpha^{q^{d-1}}$. ■

We now prove a theorem which has important consequences.

Theorem 2.40 $\mathcal{I}_{q,m}(X) = X^{q^m} - X$ is the product of all monic irreducible polynomials in $\mathbb{F}_q[X]$, whose degrees divide m.

Proof We have $\mathcal{I}_{q,m}(X) = \prod_{\alpha\in\mathbb{F}_{q^m}}(X - \alpha)$. By Corollary 2.18, the minimal polynomial $f_\alpha(X)$ of $\alpha \in \mathbb{F}_{q^m}$ over $\mathbb{F}_q$ divides $\mathcal{I}_{q,m}(X)$. By Corollary 2.16, $\deg f_\alpha$ divides m. Finally, since $f_\alpha(X) = f_\beta(X)$ or $\gcd(f_\alpha(X), f_\beta(X)) = 1$ depending on whether α and β are conjugates or not, $\mathcal{I}_{q,m}(X)$ is *a* product of monic irreducible polynomials of $\mathbb{F}_q[X]$, whose degrees divide m. In order to show that $\mathcal{I}_{q,m}(X)$ is *the* product of all such polynomials, let us consider an arbitrary polynomial $g(X) \in \mathbb{F}_q[X]$ which is

monic and irreducible over $\mathbb{F}_q$ and has degree $d|m$. The polynomial g splits over $\mathbb{F}_{q^d}$ (with no multiple roots, finite fields being perfect). Since $d|m$, by Proposition 2.34 $\mathbb{F}_{q^d} \subseteq \mathbb{F}_{q^m}$. Thus g splits over $\mathbb{F}_{q^m}$ as well and, in particular, divides $\mathcal{I}_{q,m}$. ■

The first consequence of Theorem 2.40 is that it leads to a procedure for checking the irreducibility of a polynomial $f(X) \in \mathbb{F}_q[X]$. Let $d := \deg f$. If $f(X)$ is reducible, it admits an irreducible factor of degree $\leqslant \lfloor d/2 \rfloor$. Since $g_m := \gcd(f, \mathcal{I}_{q,m})$ is the product of all distinct irreducible factors of f with degrees dividing m, we compute the gcds $g_1, \ldots, g_{\lfloor d/2 \rfloor}$. If all these gcds are 1, we conclude that f is irreducible. Otherwise f is reducible. We will see an optimized implementation of this procedure in Chapter 3. Besides irreducibility testing, the above theorem also leads to algorithms for finding random irreducible polynomials and for factorizing polynomials, as we will also discuss in Chapter 3.

The second consequence of Theorem 2.40 is that it gives us a formula for calculating the number of monic irreducible polynomials of a given degree over a given field. First we need to define a function on $\mathbb{N}$.

Definition 2.58 The *Möbius function* $\mu : \mathbb{N} \to \{0, 1, -1\}$ is defined as

$$\mu(n) := \begin{cases} 1 & \text{if } n = 1, \\ (-1)^r & \text{if } n \text{ is the product of } r \textit{ distinct} \text{ primes}, \\ 0 & \text{if } n \text{ is divisible by the square of a prime.} \end{cases}$$

It follows that $\mu(n) \neq 0$ if and only if n is square-free. □

Lemma 2.6 For $n \in \mathbb{N}$, we have

$$\sum_{d|n} \mu(d) = \begin{cases} 1 & \text{if } n = 1, \\ 0 & \text{if } n > 1, \end{cases}$$

where $\sum_{d|n}$ denotes summation over all positive divisors d of n.

Proof The result follows immediately for $n = 1$. For $n > 1$, write $n = p_1^{e_1} \cdots p_r^{e_r}$, where $p_1, \ldots, p_r$ are $r \geqslant 1$ distinct primes and $e_i \in \mathbb{N}$. The only non-zero terms in the sum $\sum_{d|n} \mu(d)$ are those corresponding to $d = p_{i_1} \cdots p_{i_s}$ for pairwise distinct choices of $i_1, \ldots, i_s \in \{1, \ldots, r\}$, $0 \leqslant s \leqslant r$. From definition, it then follows that $\sum_{d|n} \mu(d) = \sum_{s=0}^{r} \binom{r}{s} (-1)^s = (1-1)^r = 0$. ■

Lemma 2.7 Möbius inversion formula Let f and g be maps from $\mathbb{N}$ to an Abelian group G.

(1) If G is additive and $g(n) = \sum_{d|n} f(n)$, then

$$f(n) = \sum_{d|n} \mu(d) g(n/d) = \sum_{d|n} \mu(n/d) g(d).$$

(2) If G is multiplicative and $g(n) = \prod_{d|n} f(n)$, then

$$f(n) = \prod_{d|n} g(n/d)^{\mu(d)} = \prod_{d|n} g(d)^{\mu(n/d)}.$$

Proof To prove the additive formula we note that

$$\sum_{d|n}\mu(d)g(n/d) = \sum_{d|n}\Big[\mu(d)\sum_{d'|(n/d)} f(d')\Big] = \sum_{dd'|n}\mu(d)f(d')$$
$$= \sum_{d'|n}\Big[f(d')\sum_{d|(n/d')}\mu(d)\Big] = f(n),$$

where the last equality follows from Lemma 2.6. The multiplicative formula can be proved similarly. ■

Let us denote by $\nu_{q,m}$ the number of monic irreducible polynomials in $\mathbb{F}_q[X]$ of degree m and by $\mathcal{J}_{q,m}(X)$ the product of all monic irreducible polynomials in $\mathbb{F}_q[X]$ of degree m. By Theorem 2.40, we have $q^m = \sum_{d|m} d\cdot\nu_{q,d}$ and $\mathcal{I}_{q,m}(X) = X^{q^m} - X = \prod_{d|m}\mathcal{J}_{q,d}$. Applications of the Möbius inversion formula then yield the following formulas:

$$\nu_{q,m} = \frac{1}{m}\sum_{d|m}\mu(d)q^{m/d} \tag{2.4}$$

$$\mathcal{J}_{q,m} = \prod_{d|m}\mathcal{I}_{q,\frac{m}{d}}^{\mu(d)} = \prod_{d|m}\left(X^{q^{\frac{m}{d}}} - X\right)^{\mu(d)}$$

If $p_1,\ldots,p_r$ are all the prime divisors of m (not necessarily all distinct), Equation (2.4) together with the observation that $\mu(n) \geqslant -1$ for all $n \in \mathbb{N}$ imply that $\nu_{q,m} \geqslant \frac{1}{m}\left(q^m - \sum_{\substack{d|m\\ d>1}} q^{m/d}\right) \geqslant \frac{1}{m}\left(q^m - (2^r-1)q^{m/2}\right)$. But each $p_i \geqslant 2$, so that $m \geqslant 2^r$, and hence $\nu_{q,m} \geqslant \frac{1}{m}\left(q^m - (m-1)q^{m/2}\right) > 0$. We, therefore, have an independent proof of the second statement in Corollary 2.17. Moreover, for practical values of q and m we have the good approximation:

$$\nu_{q,m} \approx q^m/m\,. \tag{2.5}$$

Since the total number of monic polynomials of degree m in $F_q[X]$ is q^m, a randomly chosen monic polynomial in $\mathbb{F}_q[X]$ of degree m has an approximate probability of $1/m$ for being irreducible, that is, one expects to get an irreducible polynomial of degree m, after $\mathrm{O}(m)$ random monic polynomials are picked up from $\mathbb{F}_q[X]$. These observations have an important bearing for devising efficient algorithms for finding irreducible polynomials over finite fields. (See Chapter 3.)

The conjugates of $\alpha \in \mathbb{F}_{q^m}$ over $\mathbb{F}_q$ are α^{q^i}, $i \in \mathbb{N}$. It is interesting to look at the sum and the product of the conjugates of α. By Corollary 2.18, $f_\alpha(X) := \mathrm{minpoly}_{\alpha,\mathbb{F}_q}(X) = (X-\alpha)(X-\alpha^q)\cdots(X-\alpha^{q^{d-1}}) = X^d + a_{d-1}X^{d-1} + \cdots + a_0$ for some $d \in \mathbb{N}$. Since $f_\alpha(X) \in \mathbb{F}_q[X]$, the elements $\sum_{i=0}^{d-1}\alpha^{q^i} = -a_{d-1}$ and $\prod_{i=0}^{d-1}\alpha^{q^i} = (-1)^d a_0$ belong to $\mathbb{F}_q$. Since $\alpha^{q^d} = \alpha$, for any (positive) integral multiple δ of d, the sum $\sum_{i=0}^{\delta-1}\alpha^{q^i}$ and the product $\prod_{i=0}^{\delta-1}\alpha^{q^i}$ are elements of $\mathbb{F}_q$ too.

Definition 2.59 Let $\mathbb{F}_q \subseteq \mathbb{F}_{q^m}$, $q = p^n$, be a finite extension of finite fields and let $\alpha \in \mathbb{F}_{q^m}$. The *trace* of α over $\mathbb{F}_q$ is defined as the sum

$$\mathrm{Tr}_{\mathbb{F}_{q^m}|\mathbb{F}_q}(\alpha) := \sum_{i=0}^{m-1}\alpha^{q^i}$$

and the *norm* of α over $\mathbb{F}_q$ is defined as

$$\mathrm{N}_{\mathbb{F}_{q^m}|\mathbb{F}_q}(\alpha) := \prod_{i=0}^{m-1} \alpha^{q^i}.$$

In view of the preceding discussion, the trace and norm of α are elements of $\mathbb{F}_q$. For $q = p$, the trace and norm of α are also called the *absolute trace* and the *absolute norm* of α and are often denoted as $\mathrm{Tr}_{\mathbb{F}_{q^m}}(\alpha)$ and $\mathrm{N}_{\mathbb{F}_{q^m}}(\alpha)$. We often drop the suffix ($\mathbb{F}_{q^m}|\mathbb{F}_q$ or $\mathbb{F}_{q^m}$) in these notations, when no ambiguities are likely. □

The trace and norm functions play an important role in the theory of finite fields. See Exercise 2.86 for some elementary properties of these functions.

2.9.3 Representation of Finite Fields

$\mathbb{F}_{q^m}$ is a vector space of dimension m over $\mathbb{F}_q$. Let $\beta_0, \ldots, \beta_{m-1}$ be an $\mathbb{F}_q$-basis of $\mathbb{F}_{q^m}$. Each element $a \in \mathbb{F}_{q^m}$ has a unique representation $a = a_0\beta_0 + \cdots + a_{m-1}\beta_{m-1}$ with each $a_i \in \mathbb{F}_q$. Therefore, if we have a representation of the elements of $\mathbb{F}_q$, we can also represent the elements of $\mathbb{F}_{q^m}$. Thus elements of any finite field can be represented, if we have representations of elements of prime fields. But the set $\{0, 1, \ldots, p-1\}$ under the modulo p arithmetic represents $\mathbb{F}_p$.

So our problem reduces to selecting suitable bases $\beta_0, \ldots, \beta_{m-1}$ of $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$. In order to illustrate how we can do that, let us choose a priori a fixed monic irreducible polynomial $f(X) \in \mathbb{F}_q[X]$ with $\deg f = m$. We then represent $\mathbb{F}_{q^m} \cong \mathbb{F}_q[X]/\langle f\rangle \cong \mathbb{F}_q(\alpha)$, where α (the residue class of X) is a root of f in $\mathbb{F}_{q^m}$. The elements $1, \alpha, \ldots, \alpha^{m-1} \in \mathbb{F}_{q^m}$ are linearly independent over $\mathbb{F}_q$, since otherwise we would have a non-zero polynomial of degree less than m, of which α is a root. The $\mathbb{F}_q$-basis $1, \alpha, \ldots, \alpha^{m-1}$ of $\mathbb{F}_{q^m}$ is called a *polynomial basis* (with respect to the defining polynomial f). The elements of $\mathbb{F}_{q^m}$ are then polynomials of degrees $< m$. The arithmetic in $\mathbb{F}_{q^m}$ is carried out as the polynomial arithmetic of $\mathbb{F}_q[X]$ modulo the irreducible polynomial f.

Example 2.19

(1) The elements of $\mathbb{F}_2$ are 0 and 1 with $0+0=0, 0+1=1, 1+0=1, 1+1=0$, $0\cdot 0 = 1\cdot 0 = 0\cdot 1 = 0$ and $1\cdot 1 = 1$. In order to represent $\mathbb{F}_8 = \mathbb{F}_{2^3}$, we choose the irreducible polynomial $f(X) := X^3 + X^2 + 1 \in \mathbb{F}_2[X]$. Elements of $\mathbb{F}_8$ are $a_2\alpha^2 + a_1\alpha + a_0$, where $a_i \in \{0, 1\}$. In order to demonstrate the arithmetic in $\mathbb{F}_8$, we take $a := \alpha^2+1, b := \alpha^2+\alpha \in \mathbb{F}_8$. Their sum in $\mathbb{F}_8 = \mathbb{F}_2[\alpha]$ is $a+b = \alpha+1$. On the other hand, $ab = \alpha^4+\alpha^3+\alpha^2+\alpha = \alpha(\alpha^3+\alpha^2+1)+\alpha^2 = \alpha\cdot 0+\alpha^2 = \alpha^2$. The complete multiplication table for this representation is given in the Table 2.2.

(2) $\mathbb{F}_3$ is represented by the set $\{0, 1, 2\}$ with arithmetic operations modulo 3. Since -1 is a quadratic non-residue modulo 3, the polynomial $X^2 + 1$ is irreducible over $\mathbb{F}_3$. Therefore, the quotient field $\mathbb{F}_3[X]/\langle X^2 + 1\rangle \cong \mathbb{F}_3[\beta]$ can be used to represent $\mathbb{F}_9$, $\beta \in \mathbb{F}_3$ being a root of this polynomial. The multiplication table of $\mathbb{F}_9$ under this representation is then as shown in Table 2.3. □

Polynomial bases are most common in finite field implementations. Some other types of bases also deserve specific mention in this context.

Table 2.2 Multiplication table for $\mathbb{F}_8 = \mathbb{F}_2[X]/\langle X^3 + X^2 + 1\rangle$

	0	1	α	$\alpha+1$	α^2	α^2+1	$\alpha^2+\alpha$	$\alpha^2+\alpha+1$
0	0	0	0	0	0	0	0	0
1	0	1	α	$\alpha+1$	α^2	α^2+1	$\alpha^2+\alpha$	$\alpha^2+\alpha+1$
α	0	α	α^2	$\alpha^2+\alpha$	α^2+1	$\alpha^2+\alpha+1$	1	$\alpha+1$
$\alpha+1$	0	$\alpha+1$	$\alpha^2+\alpha$	α^2+1	1	α	$\alpha^2+\alpha+1$	α^2
α^2	0	α^2	α^2+1	1	$\alpha^2+\alpha+1$	$\alpha+1$	α	$\alpha^2+\alpha$
α^2+1	0	α^2+1	$\alpha^2+\alpha+1$	α	$\alpha+1$	$\alpha^2+\alpha$	α^2	1
$\alpha^2+\alpha$	0	$\alpha^2+\alpha$	1	$\alpha^2+\alpha+1$	α	α^2	$\alpha+1$	α^2+1
$\alpha^2+\alpha+1$	0	$\alpha^2+\alpha+1$	$\alpha+1$	α^2	$\alpha^2+\alpha$	1	α^2+1	α

Table 2.3 Multiplication table for $\mathbb{F}_9 = \mathbb{F}_3[X]/\langle X^2 + 1\rangle$

	0	1	2	β	$\beta+1$	$\beta+2$	2β	$2\beta+1$	$2\beta+2$
0	0	0	0	0	0	0	0	0	0
1	0	1	2	β	$\beta+1$	$\beta+2$	2β	$2\beta+1$	$2\beta+2$
2	0	2	1	2β	$2\beta+2$	$2\beta+1$	β	$\beta+2$	$\beta+1$
β	0	β	2β	2	$\beta+2$	$2\beta+2$	1	$\beta+1$	$2\beta+1$
$\beta+1$	0	$\beta+1$	$2\beta+2$	$\beta+2$	2β	1	$2\beta+1$	2	β
$\beta+2$	0	$\beta+2$	$2\beta+1$	$2\beta+2$	1	β	$\beta+1$	2β	2
2β	0	2β	β	1	$2\beta+1$	$\beta+1$	2	$2\beta+2$	$\beta+2$
$2\beta+1$	0	$2\beta+1$	$\beta+2$	$\beta+1$	2	2β	$2\beta+2$	β	1
$2\beta+2$	0	$2\beta+2$	$\beta+1$	$2\beta+1$	β	2	$\beta+2$	1	2β

Definition 2.60 An element $\alpha \in \mathbb{F}_{q^m}$ is called a *normal element* over $\mathbb{F}_q$, if the conjugates $\alpha, \alpha^q, \ldots, \alpha^{q^{m-1}}$ are (distinct and) linearly independent over $\mathbb{F}_q$. For a normal element α of $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$, the $\mathbb{F}_q$-basis $\alpha, \alpha^q, \ldots, \alpha^{q^{m-1}}$ is called a *normal basis* (of $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$). If, in addition, α is a primitive element (that is, a generator) of $\mathbb{F}_{q^m}^*$, then α and the corresponding normal basis are called a *primitive normal element* and a *primitive normal basis* respectively. □

It can be shown that normal bases exist for all finite extensions $\mathbb{F}_q \subseteq \mathbb{F}_{q^m}$. It can even be shown that primitive normal bases also do exist for all such extensions.

Example 2.20 Consider the representation of $\mathbb{F}_8$ in Example 2.19. The elements α, α^2 and $\alpha^4 = \alpha^2 + \alpha + 1$ satisfy

$$\begin{pmatrix} \alpha \\ \alpha^2 \\ \alpha^4 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ \alpha \\ \alpha^2 \end{pmatrix}$$

with the 3×3 transformation matrix having determinant 1 modulo 2. Thus α is a normal element of $\mathbb{F}_8$ and $(\alpha, \alpha^2, \alpha^4)$ is a normal basis of $\mathbb{F}_8$. Since $\#\mathbb{F}_8^* = 7$ is prime, α is a generator of $\mathbb{F}_8^*$, that is, α is also a primitive normal element of $\mathbb{F}_8$.

On the other hand, $\alpha+1$ is not a normal element of $\mathbb{F}_8$. Table 2.2 gives

$$\begin{pmatrix} \alpha+1 \\ (\alpha+1)^2 \\ (\alpha+1)^4 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ \alpha \\ \alpha^2 \end{pmatrix}$$

with the transformation matrix having determinant zero modulo 2. □

Computations over finite fields often call for exponentiations of elements $a = a_0\beta_0 + \cdots + a_{m-1}\beta_{m-1}$. If $\beta_i = \alpha^{q^i}$, $i = 0, \ldots, m-1$, construct a normal basis, $a^q = a_0^q\alpha^q + a_1^q\alpha^{q^2} + \cdots + a_{m-1}^q\alpha^{q^m} = a_{m-1}\alpha + a_0\alpha^q + \cdots + a_{m-2}\alpha^{q^{m-1}}$, since $\alpha^{q^m} = \alpha$ and $a_i^q = a_i$ for each i. Thus the coefficients of a^q (in the representation under the given normal basis) is obtained simply by cyclically shifting the coefficients $a_0, \ldots, a_{m-1}$ in the representation of a. This leads to a considerable saving of time. In particular, this trick becomes most meaningful for $q = 2$ (a case of high importance in cryptography).

Now that exponentiations become cheaper with normal bases, one should not let the common operations (addition and multiplication) turn significantly slower. The sum of $a = a_0\beta_0 + \cdots + a_{m-1}\beta_{m-1}$ and $b = b_0\beta_0 + \cdots + b_{m-1}\beta_{m-1}$ continues to remain as easy as in the case of a polynomial basis, namely, $a + b = (a_0 + b_0)\beta_0 + \cdots + (a_{m-1} + b_{m-1})\beta_{m-1}$, where each $a_i + b_i$ is calculated in $\mathbb{F}_q$. However, computing the product ab introduces difficulty. In particular, it requires the representation of $\beta_i\beta_j$, $0 \leqslant i, j \leqslant m-1$, in the basis $\beta_0, \ldots, \beta_{m-1}$, say, $\beta_i\beta_j = \lambda_0^{(ij)}\beta_0 + \cdots + \lambda_{m-1}^{(ij)}\beta_{m-1}$. For $i \leqslant j$, we have $\beta_i\beta_j = \alpha^{q^i} \cdot \alpha^{q^j} = \left(\alpha \cdot \alpha^{q^{j-i}}\right)^{q^i} = (\beta_0\beta_{j-i})^{q^i}$. It is thus sufficient to look only at the coefficients $\lambda_k^{(0j)}$, $0 \leqslant j, k \leqslant m-1$. We denote by C_α the number of non-zero $\lambda_k^{(0j)}$. From practical considerations (for example, for hardware implementations), C_α should be as small as possible. For $q = 2$, one can show that $2m - 1 \leqslant C_\alpha \leqslant m^2$. If, for this special case, $C_\alpha = 2m - 1$, the normal basis $\alpha, \alpha^q, \ldots, \alpha^{q^{m-1}}$ is called an *optimal normal basis*. Unlike normal (or primitive normal) bases, optimal normal bases do not exist for all values of $m \in \mathbb{N}$.

We finally mention another representation of elements of a finite field $\mathbb{F}_q$, that does not depend on the vector space representation discussed so far, but which is based on the fact that the group $\mathbb{F}_q^*$ is cyclic. If we are given a primitive element (that is, a generator) γ of $\mathbb{F}_q^*$, then the elements of $\mathbb{F}_q$ are $0, 1 = \gamma^0, \gamma, \ldots, \gamma^{q-2}$. Multiplication and exponentiation become easy with this representation, since $0 \cdot a = 0$ for all $a \in \mathbb{F}_q$, whereas $\gamma^i \cdot \gamma^j = \gamma^k$ with $k \equiv i + j \pmod{q-1}$. Unfortunately, this representation provides no clue on how to compute $\gamma^i + \gamma^j$. One possibility is to store a table consisting of the values z_k satisfying $1 + \gamma^k = \gamma^{z_k}$ for all $k = 0, \ldots, q-2$ (with $\gamma^k \neq -1$), so that for $i \leqslant j$ one can compute $\gamma^i + \gamma^j = \gamma^i(1 + \gamma^{j-i}) = \gamma^i\gamma^{z_{j-i}} = \gamma^l$, where $l \equiv i + z_{j-i} \pmod{q-1}$. Such a table is called *Zech's logarithm table*, can be maintained for small values of q and may facilitate computations in extensions $\mathbb{F}_{q^m}$. But if q is large (or more correctly if p is large, where $q = p^n$), this representation of elements of $\mathbb{F}_q$ is not practical nor often feasible. Another difficulty of this representation is that it calls for a primitive element γ. If q is large and the integer factorization of $q-1$ is not provided, there are no efficient methods known for finding such an element or even for checking if a given element is primitive.

Example 2.21 Consider the representation of $\mathbb{F}_9$ in Example 2.19. By Table 2.3, $\gamma := \beta + 1$ is a generator of $\mathbb{F}_9^*$. Table 2.4 lists the powers of γ and the Zech logarithms. □

Table 2.4 Zech's logarithm table for $\mathbb{F}_9$ with respect to $\gamma = \beta + 1$

k	γ^k	$1 + \gamma^k$	z_k
0	1	2	4
1	$\beta + 1$	$\beta + 2$	7
2	2β	$2\beta + 1$	3
3	$2\beta + 1$	$2\beta + 2$	5
4	2	0	–
5	$2\beta + 2$	2β	2
6	β	$\beta + 1$	1
7	$\beta + 2$	β	6

EXERCISE SET 2.9

2.80 Let F be a field (not necessarily finite) of characteristic $p \in \mathbb{P}$ and let $a, b \in F$. Prove that $(a + b)^p = a^p + b^p$, or, more generally, $(a + b)^{p^n} = a^{p^n} + b^{p^n}$ for all $n \in \mathbb{N}$. **[H]**

2.81 Let $p \in \mathbb{P}$, $n \in \mathbb{N}$ and $q := p^n$. Prove that:
(a) If $f(X) \in \mathbb{F}_p[X]$, then $f(X^p) = f(X)^p$.
(b) If $f(X) \in \mathbb{F}_q[X]$, then $f(X^p) = g(X)^p$ for some $g(X) \in \mathbb{F}_q[X]$.

2.82 Let $p \in \mathbb{P}$, $n, m \in \mathbb{N}$ and $q := p^n$. Let $F \subseteq K$ be an extension of finite fields with $\#F = q$ and $\#K = q^m$. Show that K is the splitting field of $X^{q^m} - X \in \mathbb{F}_q[X]$ over $\mathbb{F}_q$. **[H]**

2.83 Write the addition and multiplication tables of (some representations of) the fields $\mathbb{F}_{16}$ and $\mathbb{F}_{17}$. Use these tables to find a primitive element in each of these fields and a normal element in $\mathbb{F}_{16}$ (over $\mathbb{F}_2$).

2.84 Let K be a field (not necessarily finite or of positive characteristic).
(a) Let $f(X) \in K[X]$ be of degree 2 or 3. Prove that f is reducible in $K[X]$ if and only if f has a root in K. Deduce that $X^2 + X + 1$ and $X^3 + X + 1$ are irreducible in $\mathbb{F}_2[X]$.
(b) Let $f(X) \in K[X]$ be of degree $d \geqslant 0$. The opposite of f is the polynomial $f^{\mathrm{op}}(X) := X^d f(1/X) \in K[X]$. Show that $f(X)$ is irreducible in $K[X]$ if and only if $f^{\mathrm{op}}(X)$ is irreducible in $K[X]$. Deduce that $X^3 + X^2 + 1$ is irreducible in $\mathbb{F}_2[X]$.

2.85 In this exercise, one studies the arithmetic in the finite field $\mathbb{F}_{125} = \mathbb{F}_{5^3}$.
(a) Show that $f(X) := X^3 + 2X + 4 \in \mathbb{F}_5[X]$ is irreducible.
(b) Let us represent $\mathbb{F}_{125}$ as $\mathbb{F}_5[X]/\langle f(X)\rangle$. Call $\alpha := X + \langle f(X)\rangle \in \mathbb{F}_{125}$ and consider the elements $a := 3\alpha^2 + 2\alpha + 1$ and $b := 2\alpha^2 + 3$ in $\mathbb{F}_{125}$. Compute ab^{-1} in this representation of $\mathbb{F}_{125}$. You should compute the canonical representative of ab^{-1} in $\mathbb{F}_{125}$, that is, a polynomial in α of degree < 3 with coefficients reduced modulo 5.

2.86 Let $F \subseteq K \subseteq L$ be finite extensions of finite fields with $[L : K] = s$. Let $\alpha, \beta \in K$ and $\gamma \in L$. Prove the following assertions:
(a) $\mathrm{Tr}_{K|F}(\alpha + \beta) = \mathrm{Tr}_{K|F}(\alpha) + \mathrm{Tr}_{K|F}(\beta)$ and $\mathrm{N}_{K|F}(\alpha\beta) = \mathrm{N}_{K|F}(\alpha)\,\mathrm{N}_{K|F}(\beta)$.
(b) $\mathrm{Tr}_{L|F}(\alpha) = s\,\mathrm{Tr}_{K|F}(\alpha)$ and $\mathrm{N}_{L|F}(\alpha) = \mathrm{N}_{K|F}(\alpha)^s$.
(c) Transitivity of trace and norm
$\mathrm{Tr}_{L|F}(\gamma) = \mathrm{Tr}_{K|F}(\mathrm{Tr}_{L|K}(\gamma))$ and $\mathrm{N}_{L|F}(\gamma) = \mathrm{N}_{K|F}(\mathrm{N}_{L|K}(\gamma))$.

2.87 Let $K = \mathbb{F}_q \subseteq L = \mathbb{F}_{q^m}$ be a finite extension of finite fields. In this exercise, we treat both K and L as vector spaces over K. Show that:
(a) $\mathrm{Tr}_{L|K}$ is a surjective linear transformation $L \to K$.
(b) All the linear transformations $L \to K$ are given by $T_\alpha : L \to K$, $\beta \mapsto \mathrm{Tr}_{L|K}(\alpha\beta)$, where $\alpha \in L$. (In this notation, $\mathrm{Tr}_{L|K} = T_1$.) Moreover, for distinct elements $\alpha, \alpha' \in L$ the linear transformations T_α and $T_{\alpha'}$ are distinct.

2.88 Let K and L be as in Exercise 2.87 and let $\beta \in L$. Show that $\mathrm{Tr}_{L|K}(\beta) = 0$ if and only if $\beta = \gamma^q - \gamma$ for some $\gamma \in L$.

2.89 Let K and L be as in Exercise 2.87. Two K-bases $(\beta_0, \ldots, \beta_{m-1})$ and $(\gamma_0, \ldots, \gamma_{m-1})$ of L are called *dual* or *complementary*, if $\mathrm{Tr}_{L|K}(\beta_i\gamma_j) = \delta_{ij}$.[10] Show that every K-basis of L has a unique dual basis.

2.90 Prove that every finite extension of finite fields is Galois. **[H]**

2.91 For the extension $\mathbb{F}_q \subseteq \mathbb{F}_{q^m}$, consider the map $\mathfrak{F}_q : \mathbb{F}_{q^m} \to \mathbb{F}_{q^m}$, $\alpha \mapsto \alpha^q$.
(a) Show that $\mathfrak{F}_q$ is an $\mathbb{F}_q$-automorphism of $\mathbb{F}_{q^m}$. $\mathfrak{F}_q$ is called the *Frobenius automorphism* of $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$.
(b) Show that $\mathrm{Aut}_{\mathbb{F}_q}\mathbb{F}_{q^m}$ is cyclic of order m and with $\mathfrak{F}_q$ as a generator. **[H]**

2.92 Let $f(X) \in \mathbb{F}_q[X]$ be irreducible with $\deg f = d$. Consider the extension $\mathbb{F}_q \subseteq \mathbb{F}_{q^m}$ and let $r := \gcd(d, m)$.
(a) Show that f is irreducible in $\mathbb{F}_{q^m}[X]$ if and only if $r = 1$. **[H]**
(b) More generally, show that f factors in $\mathbb{F}_{q^m}[X]$ into a product of r irreducible polynomials each of degree d/r.

2.93 Consider the representation of $\mathbb{F}_8$ in Example 2.19. Construct the minimal polynomials over $\mathbb{F}_2$ of the elements of $\mathbb{F}_8$. **[H]**

2.94 Show that the number of (ordered) $\mathbb{F}_q$-bases of $\mathbb{F}_{q^m}$ is

$$(q^m - 1)(q^m - q)(q^m - q^2)\cdots(q^m - q^{m-1}).$$

[10] The *Kronecker delta* δ on an index set I (finite or infinite) is defined for $i, j \in I$ as:

$$\delta_{ij} := \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{if } i \neq j. \end{cases}$$

* 2.10 Affine and Projective Curves

In this section, we introduce some elementary concepts from algebraic geometry, which facilitate the treatment of elliptic and hyperelliptic curves in the next two sections. We concentrate only on plane curves, because these are the only curves we need in this book. Throughout this section, K denotes a field (finite or infinite) and $\bar{K}$ the algebraic closure of K.

2.10.1 Plane Curves

The solutions of a polynomial equation $f(X, Y) = 0$ is one of the central objects of study in algebraic geometry. For example, we know that in $\mathbb{R}^2$ the equation $X^2 + Y^2 - 1 = 0$ represents a circle with origin at $(0, 0)$ and with radius 1. When we pass to an arbitrary field, it is often not possible to visualize such *plots*, but it still makes sense to talk about the set of *solutions* of such an equation. For example, the solutions of the above circle in $\mathbb{F}_3^2$ are the four discrete points $(0, 1)$, $(0, 2)$, $(1, 0)$ and $(2, 0)$. (This solution set does not really look *round*.)

One can generalize this study by considering polynomials in n indeterminates and by investigating the simultaneous solutions of m polynomials. We, however, do not intend to be so general here and concentrate only on curves defined by a *single* polynomial equation in *two* indeterminates.

Definition 2.61 For $n \in \mathbb{N}$, the *n-dimensional affine space over K* is defined to be the set $\mathbb{A}^n(K) = K^n$ consisting of all n-tuples $(x_1, \ldots, x_n)$ with each $x_i \in K$. For $n = 2$, the affine space $\mathbb{A}^2(K)$ is also called the *affine plane over K*. For a point $P = (x_1, \ldots, x_n) \in \mathbb{A}^n(K)$, the elements $x_1, \ldots, x_n \in K$ are called the *affine coordinates* of P. The affine space $\mathbb{A}^n(\bar{K})$ over the closure $\bar{K}$ is often abbreviated as $\mathbb{A}^n$, when the field K is understood from the context. □

$\mathbb{A}^n(K)$ is an n-dimensional vector space over K. For example, the affine plane can be identified with the conventional X-Y plane.

Definition 2.62 An *affine plane (algebraic) curve C over K* is defined by a polynomial $f(X, Y) \in K[X, Y]$ and is written as $C : f(X, Y) = 0$. The set $C(K)$ of *K-rational points* on an affine plane curve $C : f(X, Y) = 0$ is the set of all points $(x, y) \in \mathbb{A}^2(K)$ satisfying $f(x, y) = 0$. □

K-rational points on a plane curve are precisely the *solutions* of the defining polynomial equation. Standard examples of affine plane curves include the straight lines given by $aX + bY + c = 0$, $a, b \in K$, not both 0, and the conic sections (circles, ellipses, parabolas and hyperbolas) given by $aX^2 + bXY + cY^2 + dX + eY + f = 0$, $a, b, c, d, e, f \in K$ with at least one of a, b, c non-zero. For $K = \mathbb{R}$, the set of K-rational points can be drawn as a graph of the polynomial equation, whereas for an arbitrary field K (in particular, for finite fields) such drawings make little or no sense. However, it is often helpful to visualize curves as curves over $\mathbb{R}$ (also called *real curves*) and then generalize the situation to an arbitrary field K.

The number ∞ is *not* treated as a real number (or integer or natural number). But it is often helpful to extend the definition of $\mathbb{R}$ by including two points that are infinitely

far away from the origin, one in each direction. This gives us the so-called *extended real line* $\hat{\mathbb{R}} := \mathbb{R} \cup \{\infty, -\infty\}$. An immediate advantage of such a *completion* of $\mathbb{R}$ is that every Cauchy sequence converges in $\hat{\mathbb{R}}$. But for studying the roots of polynomial equations it is helpful to add only a *single* point at infinity to $\mathbb{R}$ in order to get what is called the *projective line* $\mathbb{P}^1(\mathbb{R})$ over $\mathbb{R}$. Similarly, if we start with the affine plane $\mathbb{A}^2(\mathbb{R}) = \mathbb{R}^2$ and add a point at infinity for each slope $a \in \mathbb{R}$ of straight lines $Y = aX + b$ and one more for the lines $X = c$, we get the so-called *projective plane* $\mathbb{P}^2(\mathbb{R})$ over $\mathbb{R}$. We also call the *line* passing through all the points at infinity in $\mathbb{P}^2(\mathbb{R})$ to be the *line at infinity*. An immediate benefit of passing from $\mathbb{A}^2(\mathbb{R})$ to $\mathbb{P}^2(\mathbb{R})$ is that in $\mathbb{P}^2(\mathbb{R})$ any two distinct lines (parallel or not in $\mathbb{A}^2(\mathbb{R})$) meet at exactly one point and through any two distinct points of $\mathbb{P}^2(\mathbb{R})$ passes a unique line.

Now it is time to replace $\mathbb{R}$ by an arbitrary field K and rephrase our definitions in such a way that it continues to make sense to talk about points and line at infinity, even when K itself contains only finitely many points.

Definition 2.63 Let $n \in \mathbb{N}$. Define the relation $\sim$ on the 'punctured' $n+1$-dimensional affine space $\mathbb{A}^{n+1}(K) \setminus \{(0, \ldots, 0)\}$ over K by $(x_0, \ldots, x_n) \sim (y_0, \ldots, y_n)$ if and only if there exists a $\lambda \in K^*$ such that $y_i = \lambda x_i$ for all $i = 0, \ldots, n$. It is easy to see that $\sim$ is an equivalence relation on $\mathbb{A}^{n+1}(K) \setminus \{(0, \ldots, 0)\}$. The set of all equivalence classes of $\sim$ is called the *n-dimensional projective space $\mathbb{P}^n(K)$ over K*. In particular, $\mathbb{P}^2(K)$ is called the *projective plane* over K. A point $P = [x_0, \ldots, x_n] \in \mathbb{P}^n(K)$ is the equivalence class of a point $(x_0, \ldots, x_n) \in \mathbb{A}^{n+1}(K) \setminus \{(0, \ldots, 0)\}$. The elements $x_0, \ldots, x_n \in K$ constitute *a* set of *homogeneous coordinates* for P. □

It is evident that $\mathbb{P}^n(K)$ can be identified with the set of all 1-dimensional vector subspaces (that is lines) of the affine space $\mathbb{A}^{n+1}(K) = K^{n+1}$. To argue that this formal definition tallies with the intuitive notion for $n = 2$ and $K = \mathbb{R}$, consider the affine 3-space $\mathbb{R}^3$ referred to by the coordinates X, Y, Z. Look at the family of planes $\mathcal{E}_\lambda : Z = \lambda$, $\lambda \in K$, parallel to the X-Y plane. ($\mathcal{E}_0$ is the X-Y plane itself.) First take a non-zero value of λ, say $\lambda = 1$. Every line in $\mathbb{R}^3$ passing through the origin and not parallel to the X-Y plane meets $\mathcal{E}_1$ exactly at one point. Conversely, a unique line passes through each point on $\mathcal{E}_1$ and the origin. In this way, we associate points of $\mathbb{R}^2$ with points on $\mathcal{E}_1$. These are all the *finite* points of $\mathbb{R}^2$. On the other hand, the lines passing through the origin and lying in the X-Y plane ($\mathcal{E}_0 : Z = 0$) do not meet $\mathcal{E}_1$ and correspond to the points at infinity of $\mathbb{R}^2$.

In the last paragraph, we obtained the canonical embedding of the affine plane $\mathbb{A}^2(K)$ in $\mathbb{P}^2(K)$ by setting $Z = 1$. By definition, $\mathbb{P}^2(K)$ is symmetric in X, Y and Z. This means that we can as well set $X = 1$ or $Y = 1$ and see that there are other embeddings of $\mathbb{A}^2(K)$ in $\mathbb{P}^2(K)$. This observation often proves to be useful (for example, see Definition 2.66).

Now that we have passed from the affine plane to the projective plane, we should be able to carry (affine) plane curves to the projective plane. For this, we need some definitions.

Definition 2.64 Let R denote the polynomial ring $K[X_0, X_1, \ldots, X_n]$ over a field K. A *monomial* of R is an element of R of the form $X_0^{\alpha_0} X_1^{\alpha_1} \cdots X_n^{\alpha_n}$, $\alpha_i \geqslant 0$. A *term* in R is a monomial multiplied by an element $a \in K$. Any polynomial $f \in R$ is a sum of finitely many non-zero terms. The *degree* of a monomial $X_0^{\alpha_0} X_1^{\alpha_1} \cdots X_n^{\alpha_n}$ (or a term $aX_0^{\alpha_0} X_1^{\alpha_1} \cdots X_n^{\alpha_n}$)

is defined as $\alpha_0 + \alpha_1 + \cdots + \alpha_n$. The *degree* of a non-zero polynomial $f \in R$, denoted $\deg f$, is defined to be the maximum of the degrees of its non-zero terms. The degree of the zero polynomial is taken to be $-\infty$. A non-zero polynomial $f \in R$ is said to be *homogeneous* of degree $d \geqslant 0$, if all of its non-zero terms have degree d. The zero polynomial is said to be homogeneous of any degree.

Let $C : f(X, Y) = 0$ be an affine plane curve over a field K defined by a non-zero polynomial $f \in K[X, Y]$ and $d := \deg f$. Then $f^{(h)}(X, Y, Z) := Z^d f(X/Z, Y/Z)$ is a homogeneous polynomial of degree d in the polynomial ring $K[X, Y, Z]$. The polynomial $f^{(h)}$ is called the *homogenization* of f. Putting $Z = 1$ in $f^{(h)}(X, Y, Z)$ gives back the original polynomial $f(X, Y)$, that is, $f^{(h)}(X, Y, 1) = f(X, Y)$. Therefore, f is called the *dehomogenization* of the homogeneous polynomial $f^{(h)}$. The homogenization (and dehomogenization) of the zero polynomial is taken to be the zero polynomial. □

Take $[x, y, z] \in \mathbb{P}^2(K)$ and $\lambda \in K^*$. By definition, $[x, y, z] = [\lambda x, \lambda y, \lambda z]$. Since $f^{(h)}(\lambda x, \lambda y, \lambda z) = \lambda^d f^{(h)}(x, y, z) = 0$ if and only if $f^{(h)}(x, y, z) = 0$, it makes sense to talk about the *zeros* of the homogeneous polynomial $f^{(h)}$ in the projective plane $\mathbb{P}^2(K)$. This motivates us to define projective plane curves:

Definition 2.65 A *projective plane curve* C *over* K is defined by a homogeneous polynomial $h \in K[X, Y, Z]$ and is written as $C : h(X, Y, Z) = 0$. The set $C(K)$ of *K-rational points* on a projective plane curve $C : h(X, Y, Z) = 0$ is the set of all points $[x, y, z] \in \mathbb{P}^2(K)$ such that $h(x, y, z) = 0$. □

Let $C : f(X, Y) = 0$ be an affine plane curve. The projective plane curve defined by $f^{(h)}(X, Y, Z)$ is by an abuse of notation denoted also by C. The zeros of the affine curve $C : f(X, Y) = 0$ in $\mathbb{A}^2(K)$ are in one-to-one correspondence with the *finite* zeros of $C : f^{(h)}(X, Y, Z) = 0$ in $\mathbb{P}^2(K)$ (that is, zeros with $Z = 1$). The projective curve contains some more point(s), namely those at infinity, that can be obtained by putting $Z = 0$ in $f^{(h)}(X, Y, Z)$. Passage from the affine plane to the projective plane is just that: a systematic inclusion of the points at infinity.

It is often customary to write an affine plane curve as $C : f(X, Y) = g(X, Y)$ and a projective plane curve as $C : f^{(h)}(X, Y, Z) = g^{(h)}(X, Y, Z)$ with $f^{(h)}$ and $g^{(h)}$ of the same degree. The former is the same as the curve $C : f - g = 0$, and the latter the same as $C : f^{(h)} - g^{(h)} = 0$.

A homogeneous polynomial $f(X, Y, Z) \in K[X, Y, Z]$ can be viewed as the homogenization of any of the polynomials

$$f_Z(X, Y) = f(X, Y, 1),\ f_Y(X, Z) = f(X, 1, Z) \text{ and } f_X(Y, Z) = f(1, Y, Z).$$

Consider a point $P = [a, b, c]$ on the projective curve $C : f(X, Y, Z) = 0$. Since a, b and c are not all 0, P is a *finite* point on at least one of f_X, f_Y and f_Z.

2.10.2 Polynomial and Rational Functions on Plane Curves

Throughout the rest of Section 2.10 we make the following assumption:

Assumption 2.1 K is an *algebraically closed* field, that is, $\bar{K} = K$. □

Although many of the results we state now are valid for fields that are not algebraically closed, it is convenient to make this assumption in order to avoid unnecessary complications.

Let $C : f(X,Y) = 0$ be a curve defined over K. Henceforth we assume that *the polynomial $f(X,Y)$ is irreducible over K*. Though we write the affine equation for the curve for notational simplicity, we usually work with the set $C(K)$ of the K-rational points on the corresponding projective curve. We refer to the solutions of C in the affine plane $\mathbb{A}^2(K)$ as the *finite points* on the curve.

Definition 2.66 Let $P = [a,b,c]$ be a point on a curve C defined over K. We call P a *smooth* or *regular* or *non-singular* point of C, if P satisfies the following conditions.

(a) If P is a finite point (that is, if $c \neq 0$), then P is called a smooth point on C, if the partial derivatives $\partial f/\partial X$ and $\partial f/\partial Y$ do not vanish simultaneously at $(a/c, b/c)$.

(b) If P is a point at infinity (that is, if $c = 0$), then we must have $a \neq 0$ or $b \neq 0$. Assume $a \neq 0$. (The other case can be treated similarly.) Consider the polynomial $g(Y,Z) := f^{(h)}(1,Y,Z) \in K[Y,Z]$. P is a finite point on the curve $D : g(Y,Z) = 0$. P is called a smooth point on C, if $(b/a, 0)$ is a smooth point on D, that is, if $\partial g/\partial Y$ and $\partial g/\partial Z$ do not vanish simultaneously at $(b/a, 0)$.

A non-smooth point on C is also called *non-regular* or *singular*. C is called *smooth* or *regular* or *non-singular*, if all points (finite and infinite) on C are smooth. □

Now we define polynomial functions on C. For a moment, we concentrate on the affine curve, that is, only the finite points on C. Let $g, h \in K[X,Y]$ with $g - h \in \langle f \rangle$ (that is, $f|(g-h)$). Since for any point P on C we have $f(P) = 0$, it follows that $g(P) = h(P)$. This motivates us to define the following.

Definition 2.67 The ring $K[X,Y]/\langle f \rangle$ is called the *affine coordinate ring* of C and is denoted by $K[C]$. Elements of $K[C]$ are called *polynomial functions* on C. If we denote by x and y the residue classes of X and Y respectively in $K[C]$, then a polynomial function on C is given by a polynomial $g(x,y) \in K[x,y]$.[11] By our assumption, f is an irreducible polynomial; so $\langle f \rangle$ is a prime ideal of $K[X,Y]$, that is, the coordinate ring $K[C]$ is an integral domain.

The quotient field (Exercise 2.34) of $K[C]$ is called the *function field* of C and is denoted by $K(C)$. An element of $K(C)$ is of the form $g(x,y)/h(x,y)$ with $g(x,y)$, $h(x,y) \in K[C]$, $h(x,y) \neq 0$ (that is, $h(X,Y) \notin \langle f \rangle$), and is called a *rational function* on C. □

By definition, two rational functions $g_1/h_1, g_2/h_2 \in K(C)$ are equal if and only if $g_1(x,y)h_2(x,y) - g_2(x,y)h_1(x,y) = 0$ in $K[C]$ or, equivalently, if and only if $g_1(X,Y)h_2(X,Y) - g_2(X,Y)h_1(X,Y) \in \langle f \rangle \subseteq K[X,Y]$. We define addition and multiplication of rational functions by the usual rules (Exercise 2.34).

Definition 2.68 Let $P = (a,b)$ be a finite point on the curve C. Given a polynomial function $g(x,y) \in K[C]$, the *value* of g at P is defined to be $g(P) = g(a,b) \in K$. If $r \in K(C)$ is a

[11] Recall from Section 2.7 that $K[x,y]$ is the K-algebra generated by x and y. It is *not* a polynomial algebra (in general).

rational function, then r is said to be *defined* at P, if r has a representation $r = g/h$, $g, h \in K[C]$, with $h(P) \neq 0$. In that case, we define the *value* of r at P to be $r(P) := g(P)/h(P) \in K$. If r is not defined at P, it is customary to write $r(P) = \infty$. □

By definition, $K[C]$ and $K(C)$ are collections of equivalence classes. However, the value of a polynomial or a rational function on C is independent of the representatives of the equivalence classes and is, therefore, a well-defined concept.

The above definitions can be extended to the corresponding projective curve $C : f^{(h)}(X, Y, Z) = 0$. By Exercise 2.96(e), the polynomial $f^{(h)}$ is irreducible, since we assumed f to be so.

Definition 2.69 The *function field* (denoted again by $K(C)$) of the projective curve C is the set of quotients (called *rational functions*) of the form $g(X, Y, Z)/h(X, Y, Z)$, where $g, h \in K[X, Y, Z]$ are *homogeneous* of the *same degree* and $h \notin \langle f^{(h)} \rangle$. Two rational functions g_1/h_1 and g_2/h_2 are equal if and only if $g_1h_2 - g_2h_1 \in \langle f^{(h)} \rangle$.

A rational function $r \in K(C)$ is said to be *defined* at a point $P = [a, b, c]$ on C, if r has a representation g/h with $h(a, b, c) \neq 0$. In that case, we define $r(P) := g(a, b, c)/h(a, b, c)$. Since g and h are homogeneous and of the same degree, the value $r(P)$ is independent of the choice of the projective coordinates of P (Exercise 2.95). If r is not defined at P, we write $r(P) = \infty$. □

One can define polynomial functions on a projective curve (as we did for affine curves), but it makes no sense to talk about the value of such a polynomial function at a point P on the curve, because this value depends on the choice of the homogeneous coordinates of P (Exercise 2.95). This problem is eliminated for a rational function g/h by assuming g and h to be of the same degree.

Definition 2.70 Let C be a projective plane curve, r be a non-zero rational function and P a point on C. P is called a *zero* of r if $r(P) = 0$, and a *pole* of r if $r(P) = \infty$. □

Now we define the multiplicities of zeros and poles of a rational function or, more generally, the order of any point on a projective plane curve. This is based on the following result, the proof of which is long and difficult, and is omitted.

Theorem 2.41 Let C be a projective plane curve defined by an irreducible polynomial over K and P a smooth point on C. Then there exists a rational function $u_P \in K(C)$ (depending on P) with the following properties:

(i) $u_P(P) = 0$.

(ii) For any non-zero rational function $r \in K(C)$, there exist an integer d and a rational function $s \in K(C)$ having neither a zero nor a pole at P such that $r = u_P^d s$. The integer d does not depend on the choice of u_P. ■

Definition 2.71 The function u_P of the last theorem is called a *uniformizing variable* or a *uniformizing parameter* or simply a *uniformizer* of C at P. For any non-zero rational function $r \in K(C)$, the integer d with $r = u_P^d s$ is called the *order* of r at P and is denoted by $\mathrm{ord}_P(r)$. □

The connection of poles and zeros with orders is established by the following theorem which we again avoid to prove.

Theorem 2.42 P is neither a pole nor a zero of r if and only if $\mathrm{ord}_P(r) = 0$. P is a zero of r if and only if $\mathrm{ord}_P(r) > 0$. P is a pole of r if and only if $\mathrm{ord}_P(r) < 0$. ■

If P is a zero (resp. a pole) of r, the integer $\mathrm{ord}_P(r)$ (resp. $-\mathrm{ord}_P(r)$) is called the *multiplicity* of the zero (resp. pole) P.

Theorem 2.43 Let r be a rational function on the projective plane curve C defined over K. Then r has finitely many poles and zeros. Furthermore, $\sum_{P \in C(K)} \mathrm{ord}_P(r) = 0$. ■

This is one of the theorems that demand K to be algebraically closed. More explicitly, if K is *not* algebraically closed, any rational function $r \in K(C)$ continues to have only finitely many zeros and poles, but the sum of the orders of r at these points is not necessarily equal to 0. Also note that this sum, if taken over only the finite points of C, need not be 0, even when K is algebraically closed.

2.10.3 Maps Between Plane Curves

Now that we know how to define and evaluate rational functions on a curve, we are in a position to define rational maps between two curves. Let $C_1 : f_1(X, Y, Z) = 0$ and $C_2 : f_2(X, Y, Z) = 0$ be two projective plane curves defined over K by *irreducible* homogeneous polynomials $f_1, f_2 \in K[X, Y, Z]$.

Definition 2.72 A *rational map* $\varphi : C_1 \to C_2$ (defined over K) is given by rational functions $\varphi_0, \varphi_1, \varphi_2$ in $K(C_1)$ such that for each $P \in C_1(K)$ at which all of $\varphi_0(P)$, $\varphi_1(P)$ and $\varphi_2(P)$ are defined, the point $\varphi(P) := [\varphi_0(P), \varphi_1(P), \varphi_2(P)] \in C_2(K)$. One often uses the notation $\varphi = [\varphi_0, \varphi_1, \varphi_2]$.

This, however, is not the complete story. A more precise characterization of a rational map is as follows:

A rational map $\varphi = [\varphi_0, \varphi_1, \varphi_2] : C_1 \to C_2$ is said to be *defined* at $P \in C_1(K)$, if there exists a rational function $r \in K(C_1)$ (depending on P) such that $r\varphi_0$, $r\varphi_1$ and $r\varphi_2$ are all defined at P, $(r\varphi_0)(P)$, $(r\varphi_1)(P)$ and $(r\varphi_2)(P)$ are not all zero and $\varphi(P) := [(r\varphi_0)(P), (r\varphi_1)(P), (r\varphi_2)(P)] \in C_2(K)$. A rational map which is defined at every point of $C_1(K)$ is called a *morphism*.

The curves C_1 and C_2 are said to be *isomorphic* (denoted $C_1 \cong C_2$), if there exist morphisms $\varphi : C_1 \to C_2$ and $\psi : C_2 \to C_1$ such that $\psi \circ \varphi$ and $\varphi \circ \psi$ are identity maps on $C_1(K)$ and $C_2(K)$ respectively. □

Isomorphism is an equivalence relation on the set of all projective plane curves defined over K. Since two isomorphic curves share many common algebraic and geometric properties, it is of interest in algebraic geometry to study the equivalence classes (rather than the individual curves). If $C_1 \cong C_2$ and C_2 has a *simpler* representation than C_1, then studying the properties of C_2 makes our job simpler and at the same time reveals all the common properties of C_1. (See Section 2.11 for an example.)

** 2.10.4 Divisors on Plane Curves

Let a be a *symbol* and n a positive integer. We represent by na the *formal sum* $a + \cdots + a$ (n times). We also define $0a := 0$ and $-na := n(-a)$, where the symbol $-a$ satisfies

$a + (-a) = (-a) + a = 0$. For $n_1, n_2 \in \mathbb{Z}$, we define $n_1 a + n_2 a := (n_1 + n_2)a$. The set $\{na \mid n \in \mathbb{Z}\}$ under these definitions becomes an Abelian group. If we are given two symbols a, b we can analogously define formal sums $na + mb$, $n, m \in \mathbb{Z}$, and the sum of formal sums as $(n_1 a + m_1 b) + (n_2 a + m_2 b) := (n_1 + n_2)a + (m_1 + m_2)b$. With these definitions the set $\{na + mb \mid n, m \in \mathbb{Z}\}$ becomes an Abelian group. These constructions can be generalized as follows:

Definition 2.73 Given a set (not necessarily finite) of symbols a_i, $i \in I$, the set of *formal sums* of the form $\sum_{i \in I} n_i a_i$, $n_i \in \mathbb{Z}$, where $n_i = 0$ except for finitely many $i \in I$, is an Abelian group with the addition formula $\sum_{i \in I} n_i a_i + \sum_{i \in I} m_i a_i := \sum_{i \in I} (n_i + m_i) a_i$. This group is called the *free Abelian group generated by* a_i, $i \in I$. □

Now let a_i be the K-rational points on a projective plane curve C defined over K. For notational convenience, we represent by $[P]$ the symbol corresponding to the point P on C. This removes confusions in connection with elliptic curves C (See Section 2.11) for which we intend to make a distinction between $P + Q$ and $[P] + [Q]$ for two points $P, Q \in C(K)$. The former sum is again a point on C, whereas the latter is never (the symbol corresponding to) a point on C.

Definition 2.74 A formal sum $\sum_{P \in C(K)} n_P [P]$, $n_P \in \mathbb{Z}$, where $n_P = 0$ except for finitely many $P \in C(K)$, is called a *divisor* on C. The free Abelian group generated by the symbols $[P]$ for all the points $P \in C(K)$ is called the *group of divisors* of C and is denoted by $\mathrm{Div}_K(C)$ or simply by $\mathrm{Div}(C)$, when K is implicit in the context.

Let $D = \sum_{P \in C(K)} n_P [P] \in \mathrm{Div}(C)$ be a divisor. The *support* of D is defined to be the set $\{P \in C(K) \mid n_P \neq 0\} \subseteq C(K)$ and is denoted by $\mathrm{Supp}\, D$.

The *degree* of D is defined as the integer $\sum_{P \in C(K)} n_P$ and is denoted as $\deg D$. The subset $\{D \in \mathrm{Div}(C) \mid \deg D = 0\}$ of $\mathrm{Div}(C)$ is clearly a subgroup of $\mathrm{Div}(C)$. We denote this subgroup by $\mathrm{Div}^0(C)$. □

Now we define divisors of rational functions on C. Henceforth we assume that C is *smooth* (that is, smooth at all K-rational points on C).

Definition 2.75 The *divisor of a rational function* $r \in K(C)$ is defined to be the formal sum $\mathrm{Div}(r) := \sum_{P \in C(K)} \mathrm{ord}_P(r)[P]$, where $\mathrm{ord}_P(r)$ is the order of r at P (Definition 2.71). By Theorem 2.43 $\mathrm{Div}(f) \in \mathrm{Div}^0(C)$.

A divisor $D \in \mathrm{Div}(C)$ is called *principal*, if $D = \mathrm{Div}(r)$ for some rational function $r \in K(C)$. We have $\mathrm{Div}(rr') = \mathrm{Div}(r) + \mathrm{Div}(r')$ for any rational functions $r, r' \in K(C)$. It follows that the set of all principal divisors on C is a subgroup of $\mathrm{Div}(C)$ (and of $\mathrm{Div}^0(C)$ as well). We denote this subgroup by $\mathrm{Prin}_K(C)$ or simply by $\mathrm{Prin}(C)$. The quotient group $\mathrm{Div}(C)/\mathrm{Prin}(C)$ is called the *divisor class group* or the *Picard group* of C and is denoted by $\mathrm{Pic}_K(C)$ or in short by $\mathrm{Pic}(C)$. On the other hand, the quotient $\mathrm{Div}^0(C)/\mathrm{Prin}(C)$ is denoted by $\mathrm{Pic}^0_K(C)$ or $\mathrm{Pic}^0(C)$ and is called the *Jacobian* of C. Instead of $\mathrm{Pic}^0(C)$ we use the notation $\mathbb{J}_K(C)$ or $\mathbb{J}(C)$. □

Though the Jacobian $\mathbb{J}(C)$ is defined for an arbitrary smooth curve C (defined by an irreducible polynomial), it is a special class of curves called hyperelliptic curves for which it is particularly *easy* to represent and do arithmetic in the group $\mathbb{J}(C)$. This gives us yet another family of groups on which cryptographic protocols can be built.

If K is *not* algebraically closed, we need not have $\mathrm{Div}(r) \in \mathrm{Div}^0(C)$ for a rational function $r \in K(C)$. This means that in that case the group $\mathbb{J}_K(C)$ cannot be defined

in the above manner. However, since C is also a curve defined over $\bar{K}$, we can define $\mathbb{J}_{\bar{K}}(C)$ as above and call a particular subgroup of $\mathbb{J}_{\bar{K}}(C)$ as the *Jacobian* $\mathbb{J}_K(C)$ of C over K. We defer this discussion until Section 2.12.

EXERCISE SET 2.10

In this exercise set, we do *not* assume (unless otherwise stated) that K is necessarily algebraically closed.

2.95 **(a)** For homogeneous polynomials $f_1, f_2 \in K[X_1, \ldots, X_n]$ of respective degrees d_1 and d_2, prove the following assertions:
(i) If $d_1 = d_2$, then $f_1 \pm f_2$ are homogeneous polynomials of degree d_1.
(ii) The polynomial $f_1 f_2$ is homogeneous of degree $d_1 + d_2$. Conversely, if $f_1 f_2$ is homogeneous, then f_1 and f_2 are also homogeneous.
(b) A polynomial $f \in K[X_1, \ldots, X_n]$ is homogeneous of degree d if and only if it satisfies $f(\lambda X_1, \ldots, \lambda X_n) = \lambda^d f(X_1, \ldots, X_n)$ for every $\lambda \in K^*$.

2.96 In this exercise, we generalize the notion of homogenization and dehomogenization of polynomials. Let $K[X_1, \ldots, X_n]$ denote the polynomial ring in n indeterminates. Introducing another indeterminate X_0, we define the homogenization of a polynomial $f \in K[X_1, \ldots, X_n]$ as

$$f^{(h)}(X_0, X_1, \ldots, X_n) := \begin{cases} 0 & \text{if } f = 0, \\ X_0^d f(X_1/X_0, \ldots, X_n/X_0) & \text{if } \deg f = d \geqslant 0. \end{cases}$$

Prove the following assertions.
(a) $f^{(h)}$ is an element of $K[X_0, X_1, \ldots, X_n]$ and is homogeneous of degree d.
(b) $f^{(h)}(1, X_1, \ldots, X_n) = f(X_1, \ldots, X_n)$.
(c) If $\deg f = d \geqslant 0$ and f_d is the sum of all non-zero terms of degree d in f, then we have $f^{(h)}(0, X_1, \ldots, X_n) = f_d(X_1, \ldots, X_n)$.
(d) For $f, g \in K[X_1, \ldots, X_n]$, $(fg)^{(h)} = f^{(h)} g^{(h)}$. Moreover, if $g | f$, then $g^{(h)} | f^{(h)}$ and $(f/g)^{(h)} = f^{(h)}/g^{(h)}$. Under what condition(s) is $(f+g)^{(h)} = f^{(h)} + g^{(h)}$?
(e) f is irreducible if and only if $f^{(h)}$ is irreducible.

2.97 Let $C : f(X, Y) = 0$ be an affine plane curve defined by a non-zero polynomial $f \in K[X, Y]$ and $C : f^{(h)}(X, Y, Z) = 0$ the corresponding projective plane curve. Let $d := \deg f = \deg f^{(h)}$ and f_d the sum of non-zero terms of f of degree d. Show that:
(a) $f^{(h)}(X, Y, 1) = f(X, Y)$ and $f^{(h)}(X, Y, 0) = f_d(X, Y)$.
(b) $(x, y) \in \mathbb{A}^2(K)$ is a K-rational point of the affine curve if and only if $[x, y, 1] \in \mathbb{P}^2(K)$ is a K-rational point of the projective curve. More generally, let $\lambda \in K^*$. The point $(x/\lambda, y/\lambda) \in \mathbb{A}^2(K)$ is a K-rational solution of f if and only if $[x, y, \lambda]$ is a K-rational solution of $f^{(h)}$.
(c) The solutions of f at infinity are obtained by solving $f^{(h)}(X, Y, 0) = f_d(X, Y) = 0$. Conclude that the curve C can have at most d points at infinity.
(d) For $a, b \in K$, each of the curves $Y - aX = b$ and $X - aY = b$ (straight lines), and $Y - X^2 = 0$ and $X - Y^2 = 0$ (parabolas) contains only one point at infinity. The hyperbola $XY - 1 = 0$ contains two points at infinity. How many points at infinity does the hyperbola $X^2 - Y^2 - 1 = 0$ contain? The circle $X^2 + Y^2 - 1 = 0$?

(e) For $a_1, a_2, a_3, a_4, a_6 \in K$, the *elliptic curve* $Y^2 + a_1XY + a_3Y = X^3 + a_2X^2 + a_4X + a_6$ contains only one point at infinity.
(f) Let $g \in \mathbb{N}$ and $u(X), v(X) \in K[X]$ with $\deg u \leqslant g$, $\deg v = 2g+1$ and v monic. Show that the *hyperelliptic curve* $Y^2 + u(X)Y = v(X)$ has only one point at infinity.

2.98 Show that the defining polynomial of the elliptic curve in Exercise 2.97(e) is irreducible. Prove the same for the hyperelliptic curve of Exercise 2.97(f). **[H]**

2.99 Show that for an ideal $\mathfrak{a} \subseteq K[X_1, \ldots, X_n]$ the following two conditions are equivalent:

(1) $\mathfrak{a}$ is generated by a set of homogeneous polynomials.

(2) If $f = \sum_{i=0}^{d} f_i \in \mathfrak{a}$, where f_i is the sum of non-zero terms of degree i in f, then $f_i \in \mathfrak{a}$ for all $i = 0, \ldots, d$. (The polynomials f_i are called the *homogeneous components* of f.)

An ideal satisfying the above equivalent conditions is called a *homogeneous ideal*. Construct an example to demonstrate that all ideals of $K[X_1, \ldots, X_n]$ need not be homogeneous.

*2.11 Elliptic Curves

The mathematics of elliptic curves is vast and complicated. A reasonably complete understanding of elliptic curves would require a book of comparable size as this. So we plan to be rather informal while talking about elliptic curves and about their generalizations called hyperelliptic curves. Interested readers can go through the books suggested at the end of this chapter to learn more about these curves. In this section, K stands for a field (finite or infinite) and $\bar{K}$ the algebraic closure of K.

2.11.1 The Weierstrass Equation

An elliptic curve E over K is a plane curve defined by the polynomial equation

$$E : Y^2 + a_1XY + a_3Y = X^3 + a_2X^2 + a_4X + a_6\,, \quad a_i \in K, \tag{2.6}$$

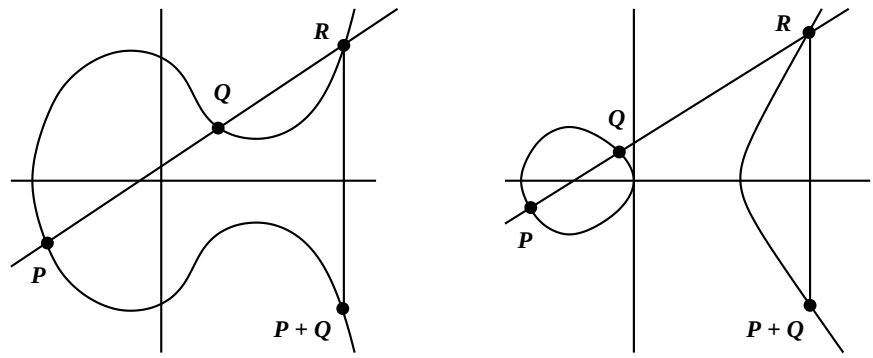
or by the corresponding homogeneous equation

$$E : Y^2Z + a_1XYZ + a_3YZ^2 = X^3 + a_2X^2Z + a_4XZ^2 + a_6Z^3\,.$$

These equations are called the *Weierstrass equations* for E. In order that E qualifies as an elliptic curve, we additionally require that it is *smooth* at all $\bar{K}$-rational points (Definition 2.66).[12] Two elliptic curves defined over the field $K = \mathbb{R}$ are shown in Figure 2.1.

E contains a single point at infinity, namely $\mathcal{O} = [0, 1, 0]$ (Exercise 2.97(e)). The set of K-rational points on E in the projective plane $\mathbb{P}^2(K)$ is denoted by $E(K)$ and

[12] *Ellipses are not elliptic curves.*



(a) **(b)**

Figure 2.1 Elliptic curves over $\mathbb{R}$ (a) $Y^2 = X^3 - X + 1$ (b) $Y^2 = X^3 - X$

is the central object of study in the theory of elliptic curves. We shortly endow $E(K)$ with a group structure and this group is used extensively in cryptography.

Let us first see how we can simplify the equation for E. The simplification depends on the characteristic of K. Because fields of characteristic 3 are only rarely used in cryptography, we will not deal with such fields. Simplification of the Weierstrass equation is effected by suitable changes of coordinates. A special kind of transformation is allowed in order to preserve the geometric and algebraic properties of an elliptic curve.

Theorem 2.44 Two elliptic curves

$$\begin{aligned} E_1 &: Y^2 + a_1XY + a_3Y = X^3 + a_2X^2 + a_4X + a_6 \\ E_2 &: Y^2 + b_1XY + b_3Y = X^3 + b_2X^2 + b_4X + b_6 \end{aligned}$$

defined over K are isomorphic (Definition 2.72) if and only if there exist $u \in K^*$ and $r, s, t \in K$ such that the substitution of $u^2X + r$ for X and $u^3Y + u^2sX + t$ for Y transforms the equation of E_1 to the equation of E_2. For this transformation, the coefficients b_i are related to the coefficients a_i as follows:

$$\left.\begin{aligned} ub_1 &= a_1 + 2s \\ u^2b_2 &= a_2 - sa_1 + 3r - s^2 \\ u^3b_3 &= a_3 + ra_1 + 2t \\ u^4b_4 &= a_4 - sa_3 + 2ra_2 - (t + rs)a_1 + 3r^2 - 2st \\ u^6b_6 &= a_6 + ra_4 + r^2a_2 + r^3 - ta_3 - t^2 - rta_1 \end{aligned}\right\} \qquad (2.7)$$

■

The theorem is not proved here. Formulas (2.7) can be checked by tedious calculations. A change of variables as in Theorem 2.44 is referred to as an *admissible change* of variables. We denote this by

$$(X, Y) \leftarrow (u^2X + r, u^3Y + u^2sX + t)\,.$$

The inverse transformation is also admissible and is given by

$$(X, Y) \leftarrow \left(\frac{X - r}{u^2}, \frac{Y - sX - t + rs}{u^3}\right).$$

Isomorphism is an equivalence relation on the set of all elliptic curves over K.

Consider the elliptic curve E over K given by Equation (2.6). If $\mathrm{char}\, K \neq 2$, the admissible change $(X, Y) \leftarrow (X, Y - \frac{a_1}{2}X - \frac{a_3}{2})$ transforms E to the form

$$E_1 : Y^2 = X^3 + b_2X^2 + b_4X + b_6.$$

If, in addition, $\mathrm{char}\, K \neq 3$, the admissible change $(X, Y) \leftarrow (\frac{X-3b_2}{36}, \frac{Y}{216})$ transforms E_1 to $E_2 : Y^2 = X^3 + aX + b$. We henceforth assume that an elliptic curve over a field of characteristic $\neq 2, 3$ is defined by

$$E : Y^2 = X^3 + aX + b, \quad a, b \in K \tag{2.8}$$

(instead of by the original Weierstrass Equation (2.6)).

If $\mathrm{char}\, K = 2$, the Weierstrass equation cannot be simplified as in Equation (2.8). In this case, we consider two cases separately, namely $a_1 \neq 0$ or otherwise. In the former case, the admissible change $(X, Y) \leftarrow \left(a_1^2X + \frac{a_3}{a_1}, a_1^3Y + \frac{a_1^2a_4+a_3^2}{a_1^3}\right)$ allows us to write Equation (2.6) in the simplified form

$$E : Y^2 + XY = X^3 + aX^2 + b\,. \tag{2.9}$$

On the other hand, if $a_1 = 0$, then the admissible change $(X, Y) \leftarrow (X + a_2, Y)$ shows that E can be written in the form

$$E : Y^2 + aY = X^3 + bX + c\,. \tag{2.10}$$

A curve defined by Equation (2.9) is called *non-supersingular*, whereas one defined by Equation (2.10) is called *supersingular*.

Now we associate two quantities with an elliptic curve. The importance of these quantities follows from the subsequent theorem. We start with the generic Weierstrass equation and later specialize to the simplified formulas.

Definition 2.76 For the curve given by Equation (2.6), we define the following quantities:

$$\begin{aligned} d_2 &= a_1^2 + 4a_2 \\ d_4 &= 2a_4 + a_1a_3 \\ d_6 &= a_3^2 + 4a_6 \\ d_8 &= a_1^2a_6 + 4a_2a_6 - a_1a_3a_4 + a_2a_3^2 - a_4^2 \\ c_4 &= d_2^2 - 24d_4 \\ \Delta(E) &= -d_2^2d_8 - 8d_4^3 - 27d_6^2 + 9d_2d_4d_6 \\ j(E) &= c_4^3/\Delta(E), \text{ if } \Delta(E) \neq 0\,. \end{aligned} \tag{2.11}$$

$\Delta(E)$ is called the *discriminant* of the curve E, and $j(E)$ the *j-invariant* of E. □

For the special cases given by the simplified equations above, these quantities have more compact formulas as given in Table 2.5.

Theorem 2.45 For the curve E defined by Equation (2.6), the following properties hold:

(1) An admissible change of variables does not alter $\Delta(E)$ and $j(E)$.

Table 2.5 Discriminant and j-invariant for elliptic curves

Special case	$\Delta(E)$	$j(E)$
$\mathrm{char}\, K \neq 2, 3$ (Equation 2.8)	$-16(4a^3 + 27b^2)$	$1728(4a)^3/\Delta(E)$
$\mathrm{char}\, K = 2$, non-supersingular (Equation 2.9)	b	$1/b$
$\mathrm{char}\, K = 2$, supersingular (Equation 2.10)	a^4	0

(2) E is an elliptic curve, that is, E is smooth, if and only if $\Delta(E) \neq 0$. In particular, the j-invariant is defined for all elliptic curves.

(3) Let E_1 and E_2 be two elliptic curves defined over the field K. If E_1 and E_2 are isomorphic over K, then $j(E_1) = j(E_2)$. Conversely, if $j(E_1) = j(E_2)$, then E_1 and E_2 are isomorphic over $\bar{K}$.

Proof (1) Tedious calculations using Formulas (2.7) establish this claim.

(2) The polynomial $f(X, Y, Z) = Y^2Z + a_1XYZ + a_3YZ^2 - X^3 - a_2X^2Z - a_4XZ^2 - a_6Z^3$ defines the curve E. Since $\partial f^{(h)}/\partial(Z)(\mathcal{O}) = 1$, E is smooth at $\mathcal{O}$. Suppose that E is not smooth at the finite point $(x_0, y_0) \in E(\bar{K})$. The admissible change $(X, Y) \leftarrow (X + x_0, Y + y_0)$ does not alter the value of $\Delta(E)$ by (1). So we can assume, without loss of generality, that $(x_0, y_0) = (0, 0)$. But then we have $f(0, 0) = -a_6 = 0$, $\partial f/\partial x(0, 0) = -a_4 = 0$ and $\partial f/\partial y(0, 0) = a_3 = 0$. Now it is easy to check from Equation (2.11) that $\Delta(E) = 0$.

Conversely, let $\Delta(E) = 0$. For simplicity, we assume that $\mathrm{char}\, K \neq 2, 3$ and E is given by Equation (2.8). By Exercise 2.62, $\Delta(X^3 + aX + b) = \frac{1}{16}\Delta(E) = 0$, that is, the polynomial $X^3 + aX + b$ has multiple roots, say, $\alpha \in \bar{K}$. But then E is not smooth at $(\alpha, 0) \in E(\bar{K})$.

(3) By Part (1) and Theorem 2.44, two isomorphic elliptic curves have the same j-invariant. For proving the converse, we once again assume that $\mathrm{char}\, K \neq 2, 3$ and $E_1 : Y^2 = X^3 + a_1X + b_1$ and $E_2 : Y^2 = X^3 + a_2X + b_2$ have the same j-invariant. Then we have $a_1^3b_2^2 = a_2^3b_1^2$. Now we provide an admissible change of variable of the form $(X, Y) \leftarrow (u^2X, u^3Y)$, $u \in \bar{K}$, that transforms E_1 to E_2. Since $\Delta(E_1) \neq 0$ and $\Delta(E_2) \neq 0$, we take $u = (b_1/b_2)^{1/6}$ if $a_1 = 0$, $u = (a_1/a_2)^{1/4}$ if $b_1 = 0$, and $u = (a_1/a_2)^{1/4} = (b_1/b_2)^{1/6}$ if $a_1b_1 \neq 0$. Note that since $\bar{K}$ is algebraically closed, u is defined in all the above cases. ■

2.11.2 The Elliptic Curve Group

Consider an elliptic curve E over a field K. We now define an operation (which is conventionally denoted by $+$) on the set $E(K)$ of K-rational points on E in the projective plane $\mathbb{P}^2(K)$. This operation provides a group structure on $E(K)$. It is important to point out that this group is not the same as the group $\mathrm{Div}_K(E)$ of divisors on $E(K)$ (Definition 2.74), since the sum of points we are going to define is *not* formal. However, there is a connection between these two groups (See Exercise 2.125).

Definition 2.77 Let E be the elliptic curve defined by Equation (2.6) and $\mathcal{O}$ the point at infinity on E. A binary operation $+$ on $E(K)$ is defined as follows:

(1) For any $P \in E(K)$, we define $\mathcal{O} + P = P + \mathcal{O} = P$, that is, $\mathcal{O}$ serves as the additive identity.

(2) The *opposite* (additive inverse) of a point $P \in E(Q)$ is now defined: if $P = \mathcal{O}$, then $-P = P$, and if $\mathcal{O} \neq P = (h, k)$, then $-P = (h, -k - a_1 h - a_3)$.

(3) For $P, Q \in E(Q) \setminus \{\mathcal{O}\}$, the sum $P + Q$ is defined by the *chord and tangent rule* which goes as follows.

(i) If $Q = -P$, then $P + Q = \mathcal{O}$.

(ii) If $Q \neq -P$, we consider the line passing through P and Q (we take the tangent line if $P = Q$). Since the degree of the defining equation for E is three, this line meets the curve at exactly one other point R. We define $P + Q = -R$. Figure 2.1 illustrates this case for curves over $\mathbb{R}$. □

Theorem 2.46 The set $E(K)$ under the operation $+$ is an Abelian group. ■

No simple proof of this theorem is known. Indeed the only group axiom that is difficult to check is associativity, that is, to check that $(P+Q)+R = P+(Q+R)$ for all $P, Q, R \in E(K)$. An elementary strategy would be to write explicit formulas for $(P+Q)+R$ and $P+(Q+R)$ (using the formulas for $P+Q$ given below) and show that they are equal, but this process involves a lot of awful calculations and consideration of many cases.

There are other proofs that are more elegant, but not as elementary. One possibility is to use the theory of divisors and is outlined now. It turns out that the Jacobian $\mathbb{J}(E)$ has a bijective correspondence with the set $E(K)$ via the map $\varphi : E(K) \to \mathbb{J}(E)$ which takes $P \in E(K)$ to $[P] - [\mathcal{O}]$ (more correctly to the equivalence class of the divisor $[P] - [\mathcal{O}]$ in $\mathbb{J}(E) = \mathrm{Div}^0(E)/\mathrm{Prin}(E)$). Furthermore, $\varphi(P + Q) = \varphi(P) + \varphi(Q)$, where the addition on the left is the addition on $E(K)$ as defined above and the addition on the right is that in the Jacobian $\mathbb{J}(E)$. By definition, $\mathbb{J}(E)$ is naturally an additive Abelian group. It immediately follows that $E(K)$ is an additive Abelian group too. (See Exercise 2.125.)

We now give the formulas for the coordinates of the points $-P$ and $P + Q$ on $E(K)$. The derivation of these formulas for the general case is left to the reader (Exercise 2.102). We concentrate on the important special cases. We assume that $P = (h_1, k_1)$ and $Q = (h_2, k_2)$ are finite points on $E(K)$ with $Q \neq -P$ so that $(h_3, k_3) := P + Q \neq \mathcal{O}$.

If $\mathrm{char}\, K \neq 2, 3$ and E is defined by Equation 2.8, we have:

$$\begin{aligned} -P &= (h_1, -k_1) \\ h_3 &= \lambda^2 - h_1 - h_2 \\ k_3 &= \lambda(h_1 - h_3) - k_1, \text{ where} \\ \lambda &= \begin{cases} \dfrac{k_2 - k_1}{h_2 - h_1}, & \text{if } P \neq Q \\ \dfrac{3h_1^2 + a}{2k_1}, & \text{if } P = Q\,. \end{cases} \end{aligned}$$

Next, we consider $\mathrm{char}\, K = 2$ and non-supersingular curves (Equation 2.9). The formulas in this case are:

$$\begin{aligned} -P &= (h_1, k_1 + h_1), \\ h_3 &= \begin{cases} \left(\frac{k_1+k_2}{h_1+h_2}\right)^2 + \frac{k_1+k_2}{h_1+h_2} + h_1 + h_2 + a, & \text{if } P \neq Q, \\ h_1^2 + \frac{b}{h_1^2}, & \text{if } P = Q, \end{cases} \\ k_3 &= \begin{cases} \left(\frac{k_1+k_2}{h_1+h_2}\right)(h_1 + h_3) + h_3 + k_1, & \text{if } P \neq Q, \\ h_1^2 + \left(h_1 + \frac{k_1}{h_1} + 1\right) h_3, & \text{if } P = Q. \end{cases} \end{aligned}$$

Finally, for supersingular curves (Equation 2.10) with $\mathrm{char}\, K = 2$, we have:

$$\begin{aligned} -P &= (h_1, k_1 + a), \\ h_3 &= \begin{cases} \left(\frac{k_1+k_2}{h_1+h_2}\right)^2 + h_1 + h_2, & \text{if } P \neq Q, \\ \frac{h_1^4 + b^2}{a^2}, & \text{if } P = Q, \end{cases} \\ k_3 &= \begin{cases} \left(\frac{k_1+k_2}{h_1+h_2}\right)(h_1 + h_3) + k_1 + a, & \text{if } P \neq Q, \\ \left(\frac{h_1^2+b}{a}\right)(h_1 + h_3) + k_1 + a, & \text{if } P = Q. \end{cases} \end{aligned}$$

We denote by mP the sum $P + \cdots + P$ (m times) for a point $P \in E(K)$ and for $m \in \mathbb{N}$. We also define $0P := \mathcal{O}$ and $(-m)P := -(mP)$ (for $m \in \mathbb{N}$).

Example 2.22 (1) Consider the elliptic curve

$$E_1 : Y^2 = X^3 + X + 3$$

over $\mathbb{F}_7$. We have $\Delta(E_1) \equiv -16(4 \times 1^3 + 27 \times 3^2) \equiv 3 \pmod 7$. Also $j(E_1) \equiv 1728 \times 4^3 \times 3^{-1} \equiv 2 \pmod 7$, that is, $j(E_1) = 2$. It is easy to check that $E_1(\mathbb{F}_7)$ contains the six points $\mathcal{O}$, $P_1 = (4, 1)$, $P_2 = (4, 6)$, $P_3 = (5, 0)$, $P_4 = (6, 1)$ and $P_5 = (6, 6)$. The multiples of these points are summarized in Table 2.6. It follows that the group $E_1(\mathbb{F}_7)$ is cyclic with P_1 as a generator.

Table 2.6 Multiples of points on the elliptic curve $Y^2 = X^3 + X + 3$ over $\mathbb{F}_7$

P	$2P$	$3P$	$4P$	$5P$	$6P$	$\mathrm{ord}\, P$
$P_0 = \mathcal{O}$						1
$P_1 = (4, 1)$	$(6, 6)$	$(5, 0)$	$(6, 1)$	$(4, 6)$	$\mathcal{O}$	6
$P_2 = (4, 6)$	$(6, 1)$	$(5, 0)$	$(6, 6)$	$(4, 1)$	$\mathcal{O}$	6
$P_3 = (5, 0)$	$\mathcal{O}$					2
$P_4 = (6, 1)$	$(6, 6)$	$\mathcal{O}$				3
$P_5 = (6, 6)$	$(6, 1)$	$\mathcal{O}$				3

(2) Now, consider the non-supersingular elliptic curve

$$E_2 : Y^2 + XY = X^3 + X^2 + \xi$$

defined over $\mathbb{F}_8 := \mathbb{F}_2[T]/\langle T^3 + T + 1\rangle = \mathbb{F}_2[\xi]$, where $\xi := T + \langle T^3 + T + 1\rangle$. We have $\Delta(E_2) = \xi$ and $j(E_2) = \xi^{-1} = \xi^2 + 1$. The finite points on E_2 are:

$$\begin{aligned}
P_1 &= (0, \xi^2 + \xi),\\
P_2 &= (1, \xi^2),\\
P_3 &= (1, \xi^2 + 1),\\
P_4 &= (\xi, \xi^2),\\
P_5 &= (\xi, \xi^2 + \xi),\\
P_6 &= (\xi + 1, \xi^2 + 1),\\
P_7 &= (\xi + 1, \xi^2 + \xi),\\
P_8 &= (\xi^2 + \xi, 1),\\
P_9 &= (\xi^2 + \xi, \xi^2 + \xi + 1).
\end{aligned}$$

So $E_2(\mathbb{F}_8)$ contains 10 points (including $P_0 := \mathcal{O}$). The multiples of the points are listed in Table 2.7, which implies that $E_2(\mathbb{F}_8)$ is again cyclic.[13] The $\phi(10) = 4$ generators of this group are P_4, P_5, P_8 and P_9.

Table 2.7 Multiples of points on the elliptic curve $Y^2 + XY = X^3 + X^2 + \xi$ over $\mathbb{F}_8$.

P	$2P$	$3P$	$4P$	$5P$	$6P$	$7P$	$8P$	$9P$	$10P$	ord P
P_0										1
P_1	$\mathcal{O}$									2
P_2	P_7	P_6	P_3	$\mathcal{O}$						5
P_3	P_6	P_7	P_2	$\mathcal{O}$						5
P_4	P_3	P_9	P_6	P_1	P_7	P_8	P_2	P_5	$\mathcal{O}$	10
P_5	P_2	P_8	P_7	P_1	P_6	P_9	P_3	P_4	$\mathcal{O}$	10
P_6	P_2	P_3	P_7	$\mathcal{O}$						5
P_7	P_3	P_2	P_6	$\mathcal{O}$						5
P_8	P_6	P_4	P_2	P_1	P_3	P_5	P_7	P_9	$\mathcal{O}$	10
P_9	P_7	P_5	P_3	P_1	P_2	P_4	P_6	P_8	$\mathcal{O}$	10

(3) Let us continue to represent $\mathbb{F}_8$ as in (2). The supersingular curve

$$E_3 : Y^2 + Y = X^3 + \xi X + \xi^2$$

has $\Delta(E_3) = 1$, $j(E_3) = 0$. $E_3(\mathbb{F}_8)$ is a cyclic group with 9 points as Table 2.8 illustrates. □

[13] Both 6 and 10 are square-free integers, and so the groups $E_1(\mathbb{F}_7)$ and $E_2(\mathbb{F}_8)$ must be cyclic (Exercise 2.115(a)).

Table 2.8 Multiples of points on the elliptic curve $Y^2 + Y = X^3 + \xi X + \xi^2$ over $\mathbb{F}_8$

P	$2P$	$3P$	$4P$	$5P$	$6P$	$7P$	$8P$	$9P$	$\operatorname{ord} P$
$P_0 = \mathcal{O}$	$\mathcal{O}$								1
$P_1 = (0, \xi^2 + \xi)$	P_5	P_4	P_7	P_8	P_3	P_6	P_2	$\mathcal{O}$	9
$P_2 = (0, \xi^2 + \xi + 1)$	P_6	P_3	P_8	P_7	P_4	P_5	P_1	$\mathcal{O}$	9
$P_3 = (\xi + 1, \xi)$	P_4	$\mathcal{O}$							3
$P_4 = (\xi + 1, \xi + 1)$	P_3	$\mathcal{O}$							3
$P_5 = (\xi^2, \xi^2)$	P_7	P_3	P_2	P_1	P_4	P_8	P_6	$\mathcal{O}$	9
$P_6 = (\xi^2, \xi^2 + 1)$	P_8	P_4	P_1	P_2	P_3	P_7	P_5	$\mathcal{O}$	9
$P_7 = (\xi^2 + \xi, \xi^2 + \xi)$	P_2	P_4	P_6	P_5	P_3	P_1	P_8	$\mathcal{O}$	9
$P_8 = (\xi^2 + \xi, \xi^2 + \xi + 1)$	P_1	P_3	P_5	P_6	P_4	P_2	P_7	$\mathcal{O}$	9

Definition 2.78 Let $m \in \mathbb{N}$. The set of points $P \in E(K)$ such that $mP = \mathcal{O}$ is evidently a subgroup of $E(K)$ and is denoted by $E_K[m]$ or by $E[m]$, if K is understood from the context. The elements of $E_K[m]$, called the *m-torsion points* of E, are those points of $E(K)$, the (additive) orders of which are finite and divide m. □

Multiples mP of a point $P \in E(K)$ can be expressed using nice formulas.

Definition 2.79 For an elliptic curve defined over K by the equation $E : f(X, Y) = 0$ and for $m \in \mathbb{N}$, there exist polynomials $\theta_m, \omega_m, \psi_m \in K[x, y] = K[X, Y]/\langle f \rangle$, such that for any point $P = (h, k) \in E(K) \setminus \{\mathcal{O}\}$ with $mP \neq \mathcal{O}$ we have

$$mP = (\theta_m(h,k)/\psi_m(h,k)^2, \omega_m(h,k)/\psi_m(h,k)^3).$$

The polynomial ψ_m is called the *m-th division polynomial* of E. □

Using the addition formula one can verify the following recursive description for ψ_m and the expressions for θ_m and ω_m in terms of ψ_m.

Lemma 2.8 For an elliptic curve E defined by the general Weierstrass Equation (2.6) over a field K, the division polynomials ψ_m, $m \in \mathbb{N}$, are recursively described as:

$$\begin{aligned}
\psi_0 &= 0 \\
\psi_1 &= 1 \\
\psi_2 &= 2y + a_1 x + a_3 \\
\psi_3 &= 3x^4 + d_2 x^3 + 3d_4 x^2 + 3d_6 x + d_8 \\
\psi_4 &= \left[2x^6 + d_2 x^5 + 5d_4 x^4 + 10 d_6 x^3 + 10 d_8 x^2 + \right. \\
&\qquad \left. (d_2 d_8 - d_4 d_6) x + d_4 d_8 - d_6^2\right] \psi_2 \\
\psi_{2m} &= \frac{(\psi_{m+2}\psi_{m-1}^2 - \psi_{m-2}\psi_{m+1}^2)\psi_m}{\psi_2} \quad \text{for } m > 2 \\
\psi_{2m+1} &= \psi_{m+2}\psi_m^3 - \psi_{m-1}\psi_{m+1}^3 \quad \text{for } m \geqslant 2,
\end{aligned}$$

where d_i are as in Definition 2.76. The polynomials θ_m satisfy

$$\theta_m = x\psi_m^2 - \psi_{m-1}\psi_{m+1} \quad \text{for all } m \in \mathbb{N},$$

and for $\mathrm{char}\, K \neq 2$, one has

$$\omega_m = \frac{\psi_{2m} - (a_1\theta_m + a_3\psi_m^2)\psi_m^2}{2\psi_m} \text{ for all } m \in \mathbb{N}. \qquad \blacksquare$$

It follows by induction on m that these formulas really give polynomial expressions for ψ_m, θ_m and ω_m for all $m \in \mathbb{N}$. For even m, the polynomial ψ_m is divisible by ψ_2. Furthermore, for $m \in \mathbb{N}$ the polynomials $\bar{f}_m(x) \in K[X,Y]/\langle f\rangle = K[x,y]$ defined as

$$\bar{f}_m(x) := \begin{cases} \psi_m(x,y)/\psi_2(x,y) & \text{if } m \text{ is even,} \\ \psi_m(x,y) & \text{if } m \text{ is odd,} \end{cases}$$

can be expressed as polynomials in x only. These univariate polynomials $\bar{f}_m$ are easier to handle than the bivariate ones ψ_m and, by an abuse of notation, are also called division polynomials. The degrees of $\bar{f}_m$ satisfy the inequality:

$$\deg \bar{f}_m \leqslant \begin{cases} (m^2-4)/2 & \text{if } m \text{ is even,} \\ (m^2-1)/2 & \text{if } m \text{ is odd.} \end{cases}$$

Points of $E[m]$ can be characterized in terms of the division polynomials:

Theorem 2.47 Let $P = (h,k) \in E(K) \setminus \{\mathcal{O}\}$ and $m \in \mathbb{N}$. Then $P \in E[m]$ if and only if $\psi_m(h,k) = 0$. Furthermore, if $m > 2$ and $2P \neq \mathcal{O}$, then $P \in E[m]$ if and only if $\bar{f}_m(h) = 0$. $\blacksquare$

We finally define polynomials $f_m \in K[x]$ as follows. If $\mathrm{char}\, K \neq 2$, then $f_m := \bar{f}_m$ for all $m \in \mathbb{N}$. On the other hand, for $\mathrm{char}\, K = 2$ and for non-supersingular curves over K we already have $\psi_m(x,y) \in K[x]$ (Exercise 2.107), and it is customary to define $f_m(x) := \psi_m(x,y)$ for all $m \in \mathbb{N}$. By further abuse of notations, we also call f_m the m-th division polynomial of E.

2.11.3 Elliptic Curves over Finite Fields

In this section, we take $K = \mathbb{F}_q$, a finite field of cardinality q and characteristic p. We do not deal with the case $p = 3$. Let E be an elliptic curve defined over $\mathbb{F}_q$. If $p > 3$, we assume that E is defined by Equation (2.8), whereas for $p = 2$, we assume that E is defined by Equation (2.10) or Equation (2.9) depending on whether E is supersingular or not.

Since $E(\mathbb{F}_q)$ is a subset of $\mathbb{P}^2(\mathbb{F}_q)$, the cardinality $\#E(\mathbb{F}_q)$ is finite. The next theorem shows that $\#E(\mathbb{F}_q)$ is quite close to q.

Theorem 2.48 **Hasse's theorem** $\#E(\mathbb{F}_q) = q+1-t$, where $|t| \leqslant 2\sqrt{q}$. (The integer t is called the *trace of Frobenius* at q.) $\blacksquare$

The implication of this theorem is that the possible cardinalities of $E(\mathbb{F}_q)$ lie in a rather narrow interval $q+1-2\sqrt{q} \leqslant \#E(\mathbb{F}_q) \leqslant q+1+2\sqrt{q}$. If $q = p$ is a prime, then for every n, $p+1-\sqrt{p} \leqslant n \leqslant p+1+\sqrt{p}$, there is at least one curve E with $\#E(\mathbb{F}_p) = n$. Moreover, the values of $\#E(\mathbb{F}_p)$ are distributed almost uniformly in the interval $p+1-\sqrt{p} \leqslant \#E(\mathbb{F}_p) \leqslant p+1+\sqrt{p}$. However, if q is not a prime, these nice results do not continue to hold.

Definition 2.80 If $t = 1$ (that is, if $\#E(\mathbb{F}_q) = q$), the curve E is called *anomalous*. If $p|t$, the curve E is called *supersingular* and if $p \not| t$, then E is called *non-supersingular*. □

Anomalous and supersingular curves are cryptographically weak, because certain algorithms are known with running time better than exponential to solve the so-called elliptic curve discrete logarithm problem over these curves. Determination of the order $\#E(\mathbb{F}_q)$ gives t from which one can easily check whether E is anomalous or supersingular. If $p = 2$, we have an easier check for supersingularity.

Proposition 2.35 An elliptic curve E over a finite field of characteristic 2 is supersingular if and only if $j(E) = 0$ or, equivalently, if and only if $a_1 = 0$ in Equation (2.6). ■

For arbitrary characteristic p, we have the following characterization.

Proposition 2.36 An elliptic curve E over $\mathbb{F}_q$ is supersingular if and only if $t^2 = 0, q, 2q, 3q$ or $4q$. In particular, if $\mathrm{char}\,\mathbb{F}_q \neq 2, 3$, then E is supersingular if and only if $t = 0$. ■

By Theorem 2.38, the group $\mathbb{F}_q^*$ is always cyclic. However, the group $E(\mathbb{F}_q)$ is not always cyclic, but is of a special kind. We need a few definitions to explain the structure of $E(\mathbb{F}_q)$. The notion of internal direct product for multiplicative groups (Exercise 2.19) can be readily applied to additive groups as follows.

Definition 2.81 Let G be an additive group and let $H_1, \ldots, H_r$ be subgroups of G. If every element of G can be written *uniquely* as $h_1 + \cdots + h_r$ with $h_i \in H_i$, $i = 1, \ldots, r$, we say that G is the (*internal*) *direct sum* of the subgroups $H_1, \ldots, H_r$ and denote this as $G = H_1 \oplus \cdots \oplus H_r$. □

Theorem 2.49 **Structure theorem for finite Abelian groups** Let G be a finite additive Abelian group of cardinality $\#G = n$. Then there exist $r \in \mathbb{N}$ and integers $n_i \geqslant 2$ for $1 \leqslant i \leqslant r$, such that G is the direct sum of (subgroups isomorphic to the) cyclic groups $\mathbb{Z}_{n_i}$, that is, $G = \mathbb{Z}_{n_1} \oplus \cdots \oplus \mathbb{Z}_{n_r}$, where $n_{i+1}|n_i$ for all $i = 1, \ldots, r-1$. Furthermore, such a decomposition is unique in the sense that if $G = \mathbb{Z}_{m_1} \oplus \cdots \oplus \mathbb{Z}_{m_s}$ with integers $m_i \geqslant 2$ and $m_{i+1}|m_i$ for $i = 1, \ldots, s-1$, then $r = s$ and $n_i = m_i$ for all $i = 1, \ldots, r$. In this case, we say that G has *rank* r and is of *type* $(n_1, \ldots, n_r)$. By Lagrange's theorem, each $n_i|n$. Moreover, $n = n_1 n_2 \cdots n_r$. G is cyclic if and only if the rank of G is 1. ■

Theorem 2.50 **Structure theorem for $E(\mathbb{F}_q)$** The elliptic curve group $E(\mathbb{F}_q)$ is of rank 1 or 2. If the rank is 1, then $E(\mathbb{F}_q)$ is cyclic, otherwise $E(\mathbb{F}_q) = \mathbb{Z}_{n_1} \oplus \mathbb{Z}_{n_2}$, where $n_1, n_2 \geqslant 2$ and $n_2|n_1$. In the second case, we have $n_2|(q-1)$. ■

Once we know the order $\#E(\mathbb{F}_q) = q + 1 - t$ of the group $E(\mathbb{F}_q)$, it is easy to compute the order of $E(\mathbb{F}_{q^m})$ as the following theorem suggests.

Theorem 2.51 Let $\alpha, \beta \in \mathbb{C}$ satisfy $1 - tX + qX^2 = (1 - \alpha X)(1 - \beta X)$. Then for any $m \in \mathbb{N}$ the order $\#E(\mathbb{F}_{q^m}) = q^m + 1 - (\alpha^m + \beta^m)$. ■

EXERCISE SET 2.11

2.100 Show that the following curves over K are not smooth (and hence not elliptic curves):

(a) $Y^2 = X^3$, K arbitrary.
(b) $Y^2 = X^3 + X^2$, K arbitrary.
(c) $Y^2 = X^3 + aX + b$, if $\mathrm{char}\, K = 2$.

2.101 **(a)** Show that for an elliptic curve E over K and a finite point $P = (h, k) \in E(K)$, the only points in $E(K)$ (or $E(\bar{K})$) having X-coordinate equal to h are P and $-P$.
(b) Let $\mathrm{char}\, K \neq 2, 3$ and let E be defined by Equation (2.8). If $\alpha_1, \alpha_2, \alpha_3 \in \bar{K}$ are the roots (distinct by Theorem 2.45) of $X^3 + aX + b$, then $(\alpha_1, 0)$, $(\alpha_2, 0)$ and $(\alpha_3, 0)$ are the only points on $E(\bar{K})$ with Y-coordinate equal to 0. Show that these are the only points of order 2 in $E(\bar{K})$.

2.102 Let $P = (h_1, k_1)$ and $Q = (h_2, k_2)$ be two points (different from $\mathcal{O}$) in $E(K)$ defined by the Weierstrass Equation (2.6). Assume that $Q \neq -P$. Determine $R = (h_3, k_3) = P + Q$ as follows:
(a) Show that the line passing through P and Q (the tangent, if $P = Q$) has the equation $Y = \lambda X + \mu$, where

$$\begin{aligned} \lambda &= \begin{cases} \dfrac{k_2 - k_1}{h_2 - h_1}, & \text{if } P \neq Q, \\ \dfrac{3h_1^2 + 2a_2h_1 + a_4 - a_1k_1}{2k_1 + a_1h_1 + a_3}, & \text{if } P = Q, \text{ and} \end{cases} \\ \mu &= k_1 - \lambda h_1 \, . \end{aligned}$$

(b) Substituting $\lambda X + \mu$ for Y in Equation (2.6) gives a cubic equation in X of which h_1 and h_2 are two roots. Show that the third root (the X-coordinate of R) is

$$h_3 = \lambda^2 + a_1\lambda - a_2 - h_1 - h_2.$$

Hence deduce that the Y-coordinate of R is

$$k_3 = -(\lambda + a_1)h_3 - \mu - a_3.$$

2.103 Let $\bar{\jmath} \in K$. Show that there exists an elliptic curve E over K such that $j(E) = \bar{\jmath}$. **[H]**

2.104 Assume that $\mathrm{char}\, K \neq 2, 3$ and consider the elliptic curve E given by Equation (2.8). Let $K[E]$ be the affine coordinate ring and $K(E)$ the field of rational functions on E.
(a) Show that every element in $K[E]$ can be uniquely represented as $u(x) + yv(x)$ for polynomials $u(x), v(x) \in K[x]$.
(b) The *conjugate* of $f = u(x) + yv(x) \in K[E]$ is defined as $\hat{f} := u(x) - yv(x)$. The *norm* of f is defined as $\mathrm{N}(f) = f\hat{f}$. Show that $\mathrm{N}(f) \in K[x]$.
(c) The *degree* of $f = u(x) + yv(x) \in K[E]$ is defined as $\deg f := \max(2 \deg_x u, 3 + 2 \deg_x v)$, where $\deg_x$ denotes the degree in x. Show that $\deg f = \deg_x \mathrm{N}(f)$.
(d) Show that for $f, g \in K[E]$ one has $\mathrm{N}(fg) = \mathrm{N}(f)\,\mathrm{N}(g)$. Hence conclude that $\deg(fg) = \deg f + \deg g$.
(e) Show that every rational function in $K(E)$ can be represented as $a(x) + yb(x)$, where $a(x), b(x) \in K(x)$.

2.105 Show that the division polynomials $\bar{f}_m$ for the general Weierstrass equation can be recursively defined as

$$\begin{aligned}
\bar{f}_0 &= 0, \quad \bar{f}_1 = 1, \quad \bar{f}_2 = 1, \quad \bar{f}_3 = \psi_3, \quad \bar{f}_4 = \psi_4/\psi_2, \\
\bar{f}_{2m} &= (\bar{f}_{m+2}\bar{f}_{m-1}^2 - \bar{f}_{m-2}\bar{f}_{m+1}^2)\bar{f}_m \quad (\text{for } m > 2), \\
\bar{f}_{2m+1} &= \begin{cases} F^2\bar{f}_{m+2}\bar{f}_m^3 - \bar{f}_{m-1}\bar{f}_{m+1}^3 & \text{if } m \text{ is even} \\ \bar{f}_{m+2}\bar{f}_m^3 - F^2\bar{f}_{m-1}\bar{f}_{m+1}^3 & \text{if } m \text{ is odd} \end{cases} \quad (\text{for } m \geqslant 2),
\end{aligned}$$

where $F = 4x^3 + d_2x^2 + 2d_4x + d_6$.

2.106 Write the recursive formulas for the division polynomials $\psi_m(x, y)$ and $\bar{f}_m$ for the elliptic curve E defined by Equation 2.8 over a field K of characteristic $\neq 2, 3$. Show that for $m \geqslant 2$ and for $P = (h, k) \in E(K) \setminus E[m]$ we have

$$mP = \left(h - \frac{\psi_{m-1}(h,k)\psi_{m+1}(h,k)}{\psi_m^2(h,k)}, \frac{\psi_{m+2}(h,k)\psi_{m-1}^2(h,k) - \psi_{m-2}(h,k)\psi_{m+1}^2(h,k)}{4k\psi_m^3(h,k)}\right).$$

2.107 Write the recursive formulas for the division polynomials $\psi_m(x, y)$ and $\bar{f}_m$ for the elliptic curve E defined by Equation 2.9 over a field K of characteristic 2. Conclude that ψ_m are polynomials in only x for all $m \in \mathbb{N}$. With $f_m := \psi_m$ for all $m \in \mathbb{N}$ show that for $m \geqslant 2$ and for $P = (h, k) \in E(K) \setminus E[m]$ we have

$$mP = \left(h + \frac{f_{m-1}(h)f_{m+1}(h)}{f_m^2(h)}, h + k + \frac{(h^2+h+k)f_{m-1}(h)f_m(h)f_{m+1}(h) + f_{m-2}(h)f_{m+1}^2(h)}{hf_m^3(h)}\right).$$

2.108 Consider the elliptic curve defined over the field $\mathbb{F}_7$:

$$E_{a,b} : Y^2 = X^3 + aX + b.$$

Verify the following assertions: (You may write a computer program.)

(a) Each $E_{a,b}$ has order between 3 and 13.
(b) The curve $E_{0,3} : Y^2 = X^3 + 3$ has the maximum possible order 13.
(c) The curve $E_{0,4} : Y^2 = X^3 + 4$ has the minimum possible order 3.
(d) The curve $E_{0,5} : Y^2 = X^3 + 5$ is anomalous.
(e) The group $E_{0,6}(\mathbb{F}_7)$ is not cyclic.

2.109 Consider the representation of $\mathbb{F}_8$ as $\mathbb{F}_2[\xi]$, where ξ is a root of T^3+T+1 in $\bar{\mathbb{F}}_2$. Identify an element $a_2\xi^2 + a_1\xi + a_0 \in \mathbb{F}_8$ (where $a_i \in \{0, 1\}$) with the integer $(a_2a_1a_0)_2 = a_2 2^2 + a_1 2 + a_0$. For integers $a, b \in \{0, 1, \ldots, 7\}$, $b \neq 0$, define the non-supersingular elliptic curve:

$$E_{a,b} : Y^2 + XY = X^3 + aX^2 + b.$$

Verify the following assertions: (You may write a computer program.)

(a) Each $E_{a,b}$ has order between 4 and 14.
(b) The curve $E_{1,1} : Y^2 + XY = X^3 + X^2 + 1$ has the maximum possible order 14.
(c) The curve $E_{2,1} : Y^2 + XY = X^3 + \xi X^2 + 1$ has the minimum possible order 4.

(d) The curve $E_{2,2} : Y^2 + XY = X^3 + \xi X^2 + \xi$ is anomalous.
(e) The orders of $E_{a,b}$ for all choices of a, b lie in the set $\{4, 6, 8, 10, 12, 14\}$.
(f) Each $E_{a,b}(\mathbb{F}_8)$ is cyclic.
(g) Theorem 2.45(3) requires the phrase *over* $\bar{K}$, that is, two curves over an algebraically non-closed field having the same j-invariant may be non-isomorphic.

2.110 Consider the representation of $\mathbb{F}_8$ and the identification of elements of $\mathbb{F}_8$ with integers as in Exercise 2.109. For $a, b, c \in \{0, 1, \ldots, 7\}$, $a \neq 0$, define the supersingular elliptic curve:

$$E_{a,b,c} : Y^2 + aY = X^3 + bX + c.$$

Verify the following assertions: (You may write a computer program.)
(a) Each $E_{a,b,c}$ has order between 5 and 13.
(b) The curve $E_{1,1,1} : Y^2 + Y = X^3 + X + 1$ has the maximum possible order 13.
(c) The curve $E_{1,1,2} : Y^2 + Y = X^3 + X + \xi$ has the minimum possible order 5.
(d) The orders of $E_{a,b,c}$ for all choices of a, b, c lie in the set $\{5, 9, 13\}$.
(e) No $E_{a,b,c}$ is anomalous.
(f) Each $E_{a,b,c}(\mathbb{F}_8)$ is cyclic.

2.111 Consider the elliptic curve $E : Y^2 + XY = X^3 + X^2 + 1$ defined over $\mathbb{F}_{2^n}$ for all $n \in \mathbb{N}$. Show that

$$\#E(\mathbb{F}_{2^n}) = 2^n + 1 - \frac{1}{2^{n-1}} \left[1 - \binom{n}{2} 7 + \binom{n}{4} 7^2 - + \cdots + (-1)^r \binom{n}{2r} 7^r \right],$$

where $r = \lfloor n/2 \rfloor$. **[H]** Conclude that E is anomalous over $\mathbb{F}_2$, but not so over $\mathbb{F}_4$.

2.112 Let K be a finite field of characteristic $\neq 2, 3$ and $E : Y^2 = X^3 + aX + b$ an elliptic curve defined over K. Prove that:
(a) $\#E(K)$ is odd if and only if $X^3 + aX + b$ is irreducible in $K[X]$. **[H]**
(b) $E(K)$ is not cyclic if $X^3 + aX + b$ splits in $K[X]$.
(c) The converse of Part (b) does not hold. **[H]**

2.113 Let $E : Y^2 + XY = X^3 + aX^2 + b$ be a non-supersingular elliptic curve defined over $\mathbb{F}_{2^m}$. Prove that:
(a) $E(\mathbb{F}_{2^m})$ has exactly one point of order 2. **[H]**
(b) $\#E(\mathbb{F}_{2^m})$ is even.

2.114 Let $E : Y^2 + aY = X^3 + bX + c$ be a supersingular elliptic curve over $\mathbb{F}_{2^m}$. Prove that:
(a) $E(\mathbb{F}_{2^m})$ has no points of order 2.
(b) $\#E(\mathbb{F}_{2^m})$ is odd.

2.115 **(a)** Let G be a finite Abelian group of cardinality n. Show that if n is square-free, then G is cyclic. **[H]**
(b) Prove that if E is an anomalous elliptic curve over $\mathbb{F}_q$, then $E(\mathbb{F}_q)$ is cyclic. **[H]**
(c) If E is a supersingular elliptic curve over the field $\mathbb{F}_q$ of characteristic $\neq 2, 3$, prove that $E(\mathbb{F}_q)$ is either cyclic or isomorphic to $\mathbb{Z}_r \oplus \mathbb{Z}_2$. **[H]**

2.116 Let $p \in \mathbb{P}$, $p \equiv 3 \pmod 4$, and $a, n \in \mathbb{N}$. Consider the elliptic curve $E : Y^2 = X^3 - a^2X$ over $\mathbb{F}_p$ (or over $\mathbb{F}_{p^n}$). Prove that:
(a) $E(\mathbb{F}_p)$ contains at most three points of order three.
(b) The points of order three in $E(\mathbb{F}_{p^n})$ are precisely the points of order three in $E(\mathbb{F}_p)$.

2.117 A Weierstrass equation of an elliptic curve defined over a field K is said to be in the *Legendre form*, if it can be written as

$$E : Y^2 = X(X-1)(X-k) \tag{2.12}$$

for some $k \in \bar{K}$, $k \neq 0, 1$. Show that if $\operatorname{char} K \neq 2$, then every Weierstrass equation over K can be written in the Legendre form. Show that the j-invariant of the curve E defined by Equation (2.12) is $2^8 \dfrac{(k^2-k+1)^3}{k^2(k-1)^2}$.

** 2.12 Hyperelliptic Curves

Hyperelliptic curves are generalizations of elliptic curves. We cannot define a group structure on a general hyperelliptic curve in the way as we did for elliptic curves. We instead work in the Jacobian of a hyperelliptic curve. For an elliptic curve E over an algebraically closed field K, the Jacobian $\mathbb{J}_K(E)$ is canonically isomorphic to the group $E(K)$. Thus one can as well use the techniques for hyperelliptic curves for describing and working in elliptic curve groups. However, the exposition of the previous section turns out to be more intuitive and computationally oriented.

2.12.1 The Defining Equations

A hyperelliptic curve C of *genus* $g \in \mathbb{N}$ over a field K is defined by a polynomial equation of the form

$$C : Y^2 + u(X)Y = v(X) \qquad \text{with } u, v \in K[X], v \text{ monic}, \ \deg u \leqslant g, \deg v = 2g+1\,. \tag{2.13}$$

In order that C qualifies as a hyperelliptic curve, we additionally require that C (as a projective curve) be smooth over $\bar{K}$. The set of K-rational points on C is denoted as usual by $C(K)$. For $g = 1$, Equation (2.13) is the same as the Weierstrass Equation (2.6) on p 98, that is, *elliptic curves are hyperelliptic curves of genus one*. A hyperelliptic curve of genus 2 over $\mathbb{R}$ is shown in Figure 2.2.

A hyperelliptic curve has only one point at infinity $\mathcal{O} = [0, 1, 0]$ (Exercise 2.97(f)) and is smooth at $\mathcal{O}$. If $\operatorname{char} K \neq 2$, substituting $(X, Y) \leftarrow (X, Y - \frac{u(X)}{2})$ simplifies Equation (2.13) as $Y^2 = v(X) + \frac{u(X)^2}{4}$. Since $v + \frac{u^2}{4}$ is a monic polynomial in $K[X]$ of degree $2g+1$, we may assume that if $\operatorname{char} K \neq 2$, the equation for C is of the form:

$$C : Y^2 = v(X) \quad \text{with } v \in K[X], v \text{ monic}, \deg v = 2g+1\,. \tag{2.14}$$

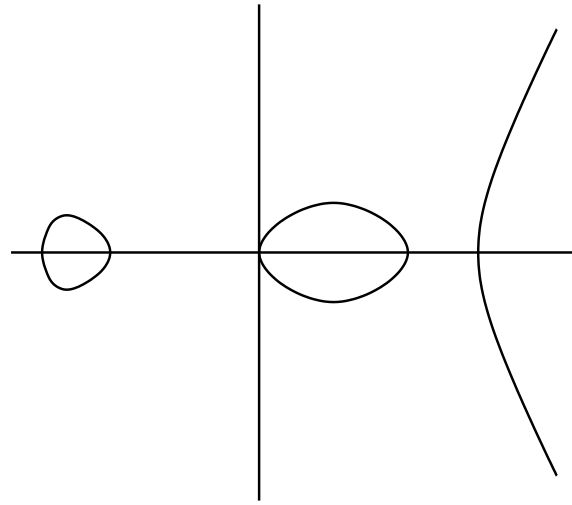


Figure 2.2 A hyperelliptic curve of genus 2 over $\mathbb{R}$: $Y^2 = X(X^2-1)(X^2-2)$

Proposition 2.37 If $\mathrm{char}\, K \neq 2$, then the hyperelliptic curve C defined by Equation (2.14) is smooth if and only if v has no multiple roots (in $\bar{K}$). If $\mathrm{char}\, K = 2$, then the curve defined by Equation (2.14) is never smooth.

Proof First, consider $\mathrm{char}\, K \neq 2$. If v has a multiple root, say $\alpha \in \bar{K}$, then $v'(\alpha) = 0$ and, therefore, C is not smooth at the finite point $(\alpha, 0) \in C(\bar{K})$. Conversely, if (h,k) is a singular point on $C(\bar{K})$, then we have $2k = 0$ and $v'(h) = 0$. Since $(h,k) = (h,0)$ is a point on C, we have $v(h) = 0$, that is, h is a multiple root of v.

For $\mathrm{char}\, K = 2$ and $(h,k) \in C(\bar{K})$, we have $(\partial(Y^2 - v(X))/\partial X)(h,k) = v'(h)$ and $(\partial(Y^2 - v(X))/\partial Y)(h,k) = 0$. Now, $v'(X)$ is a monic polynomial of degree $2g > 0$ and, therefore, has at least one root, say $\alpha \in \bar{K}$. But then C is not smooth at $(\alpha, \sqrt{v(\alpha)}) \in C(\bar{K})$. ■

Definition 2.82 Let $P = (h,k)$ be a finite point on the hyperelliptic curve C defined by Equation (2.13). The point $\tilde{P} = (h, -k - u(h)) \in C(K)$ is called the *opposite* of P.[14] P and $\tilde{P}$ are the only points on C with X-coordinate equal to h. If $P = \tilde{P}$, then P is called a *special point* on C, otherwise it is called an *ordinary point* on C. The set of all finite (resp. ordinary, resp. special) points on C is denoted by $C_{\mathrm{fin}}(K)$ (resp. $C_{\mathrm{ord}}(K)$, resp. $C_{\mathrm{spl}}(K)$). These notations are also abbreviated as C_{fin}, C_{ord} and C_{spl}, if the field K is understood from the context. □

2.12.2 Polynomial and Rational Functions

All the general theory we described in Section 2.10 continues to be valid for hyperelliptic curves. However, since we are now given an explicit equation describing the

[14] It is customary to define the opposite of $\mathcal{O}$ to be $\mathcal{O}$ itself.

curves, we can give more explicit expressions for polynomial and rational functions on hyperelliptic curves. For simplicity, we consider the affine equation and extend our definitions separately for the point at infinity.

Consider the hyperelliptic curve C defined by Equation (2.13). By Exercise 2.98, the defining polynomial $f(X,Y) := Y^2 + u(X)Y - v(X)$ (or its homogenization) is irreducible over $\bar{K}$, so that the affine (or projective) coordinate ring of C is an integral domain and the corresponding function field is simply the field of fractions of the coordinate ring.

Let $G(x,y) \in K[C]$. Since $y^2 + u(x)y - v(x) = 0$ in $K[C]$, we can repeatedly substitute y^2 by $-u(x)y + v(x)$ in $G(x,y)$ until the y-degree of $G(x,y)$ becomes less than 2. This proves part of the following:

Proposition 2.38 Every polynomial function $G(x,y) \in K[C]$ can be written uniquely as $G(x,y) = a(x) + yb(x)$ for some $a(X), b(X) \in K[X]$.

Proof In order to establish the uniqueness, note that if $G(x,y) = a_1(x) + yb_1(x) = a_2(x) + yb_2(x)$, then $[a_1(X) + Yb_1(X)] - [a_2(X) + Yb_2(X)] \in \langle f(X,Y)\rangle \subseteq K[X,Y]$. Since the Y-degree of f is 2, this implies $[a_1(X) + Yb_1(X)] - [a_2(X) + Yb_2(X)] = 0$, that is, $[a_1(X) - a_2(X)] + [b_1(X) - b_2(X)]Y = 0$, that is, $a_1(X) = a_2(X)$ and $b_1(X) = b_2(X)$. ■

Definition 2.83 Let $G(x,y) = a(x) + yb(x) \in K[C]$. The *conjugate* of G is defined to be the polynomial function $\hat{G}(x,y) := a(x) - b(x)(u(x) + y)$. The *norm* of G is defined as $\mathrm{N}(G) := G\hat{G}$. □

Some useful properties of the norm function are listed in the following lemma, the proof of which is left to the reader as an easy exercise.

Lemma 2.9 For $G, H \in K[C]$, we have:

(1) $\hat{\hat{G}} = G$.
(2) If $G(x,y) = a(x) + yb(x)$, then $\mathrm{N}(G) = a(x)^2 - a(x)b(x)u(x) - v(x)b(x)^2$. In particular, $\mathrm{N}(G) \in K[x]$.
(3) $\mathrm{N}(G) = \mathrm{N}(\hat{G})$.
(4) $\mathrm{N}(GH) = \mathrm{N}(G)\,\mathrm{N}(H)$. ■

We also have an easy description of the rational functions on C.

Proposition 2.39 Every rational function $r(x,y) \in K(C)$ can be written in the form $s(x) + yt(x)$ for some $s(X), t(X) \in K(X)$.

Proof We can write $r(x,y) = G(x,y)/H(x,y)$ for $G, H \in K[C]$, $H \neq 0$. Multiplying both the numerator and the denominator by $\hat{H}$ and using Lemma 2.9(2) and Proposition 2.38 completes the proof. ■

The *value* of a rational function on C at a *finite* point on C can be defined as in the case of general curves (See Definition 2.68). In order to define the value of a rational function at the point $\mathcal{O}$, we need some other concepts.

For a moment, let us assume that $K = \mathbb{R}$. From the equation of C, we see that $k^2 \approx h^{2g+1}$ (neglecting lower-degree terms) for sufficiently large coordinates h, k of

a point $(h,k) \in C(K)$. This means that k tends to infinity exponentially $(2g+1)/2$ times as fast as h does. So it is customary to give Y a *weight* $(2g+1)/2$ times a weight we give to X. The smallest integral weights of X and Y to satisfy this are 2 and $2g+1$ respectively. This motivates us to provide Definition 2.84 (generalized for any K).

Definition 2.84 Let $0 \neq G(x,y) = a(x) + yb(x) \in K[C]$. The *degree* of G is defined to be $\deg G := \max(2\deg_x a, 2g+1+2\deg_x b)$, where $\deg_x$ denotes the usual x-degree of a polynomial in $K[x]$. Since a and b are *uniquely* determined by G, $\deg G$ is well-defined. If $G=0$, we set $\deg G := -\infty$.

If $0 \neq G = a(x)+yb(x)$, $d_1 = \deg_x a$ and $d_2 = \deg_x b$, then the *leading coefficient* of G is taken to be the coefficient of x^{d_1} in $a(x)$ if $\deg G = 2d_1$, or to be the coefficient of x^{d_2} in $b(x)$ if $\deg G = 2g+1+2d_2$. (We cannot have $2d_1 = 2g+1+2d_2$, since the left side is even and the right side is odd.) □

Some basic properties of the degree function follow.

Lemma 2.10 For $G, H \in K[C]$, we have:

(1) $\deg G = \deg_x(\mathrm{N}(G))$.

(2) $\deg(GH) = \deg G + \deg H$.

(3) $\deg \hat{G} = \deg G$.

Proof Easy exercise. ■

Now we are in a position to give an explicit definition of the value of a rational function at $\mathcal{O}$.

Definition 2.85 For $r(x,y) = G(x,y)/H(x,y) \in K(C)$ with $G, H \in K[C]$, we define $r(\mathcal{O})$ as:
If $\deg(G) < \deg(H)$, then $r(\mathcal{O}) := 0$.
If $\deg(G) > \deg(H)$, then $r(\mathcal{O}) := \infty$ (that is, r is not defined at $\mathcal{O}$).
If $\deg(G) = \deg(H)$, then $r(\mathcal{O})$ is defined as the ratio of the leading coefficients of G and H. □

Now that we have a complete description of the value of a rational function at any point on C, poles and zeros of rational functions on C can be defined as in Definition 2.70. In order to define the order of a polynomial or rational function at a point P on C, we should find a uniformizing parameter u_P at P. Tedious calculations help one deduce the following explicit expressions for u_P.

Proposition 2.40 Let $P = (h,k) \in C(\bar{K})$ be a finite point. Then we can take

$$u_P = \begin{cases} x-h & \text{if } P \text{ is an ordinary point} \\ y-k & \text{if } P \text{ is a special point} \end{cases}$$

as a uniformizing parameter at P. Finally, $u_{\mathcal{O}} = x^g/y$ is a uniformizing parameter at the point $\mathcal{O}$ at infinity (where g is the genus of C). ■

We give an alternative definition of the order (independent of u_P), which is computationally useful and which is equivalent to Definition 2.71 for a hyperelliptic curve.

Definition 2.86 Let $G = a(x) + yb(x) \in \bar{K}[C]$ and $P \in C(\bar{K})$. The *order of G at P* is defined as follows. First, let $P = (h,k)$ be a *finite* point on C. Let e be the largest exponent

such that $(x-h)^e$ divides both $a(x)$ and $b(x)$. We write $G = (x-h)^e G_1(x,y)$. If $G_1(h,k) \neq 0$ we set $l := 0$, otherwise we set l to be the highest exponent such that $(x-h)^l$ divides $\mathrm{N}(G_1)$. We then define

$$\mathrm{ord}_P(G) := \begin{cases} e+l, & \text{if } P \in C_{\mathrm{ord}}, \\ 2e+l, & \text{if } P \in C_{\mathrm{spl}}. \end{cases}$$

Finally, we define $\mathrm{ord}_{\mathcal{O}}(G) := -\max(2\deg_x a, 2g+1+2\deg_x b)$.

Now, let $r(x,y) = G(x,y)/H(x,y)$ be a rational function on C and $P \in C(\bar{K})$. We define the *order of* r *at* P as $\mathrm{ord}_P(r) := \mathrm{ord}_P(G) - \mathrm{ord}_P(H)$. The value $\mathrm{ord}_P(r)$ can be shown to be independent of the choice of G and H. □

Example 2.23 Let $P = (h,k) \in C(K)$ be a finite point on C. Consider the rational function $r = (x-h)^m \in K(C)$, $m \in \mathbb{N}_0$. The only points on C with X-coordinate equal to h are P and its opposite $\tilde{P}$. Therefore, if P is an ordinary point, $\mathrm{ord}_P(r) = \mathrm{ord}_{\tilde{P}}(r) = m$, whereas if P is a special point, $\mathrm{ord}_P(r) = 2m$. Moreover, $\mathrm{ord}_{\mathcal{O}}(r) = -2m$. For any $Q \in C(K) \setminus \{P, \tilde{P}, \mathcal{O}\}$, we have $\mathrm{ord}_Q(r) = 0$.

Now consider $r = (x-h)^m$ for some $m < 0$. Write $r = G/H$ with $G = 1$ and $H = (x-h)^{-m}$. Since $\mathrm{ord}_Q(r) = \mathrm{ord}_Q(G) - \mathrm{ord}_Q(h)$, we continue to have

$$\mathrm{ord}_Q(r) = \begin{cases} m & \text{if } Q = P \text{ or } Q = \tilde{P}, \text{ where } P \neq \tilde{P}, \\ 2m & \text{if } Q = P = \tilde{P}, \\ -2m & \text{if } Q = \mathcal{O}, \\ 0 & \text{if } Q \in C(K) \setminus \{P, \tilde{P}, \mathcal{O}\}. \end{cases}$$

If $m \geqslant 0$, then r is a polynomial function and has zeros P and $\tilde{P}$ and no finite poles. In this case, the sum of the orders of its zeros is $2m = 2\deg_x r = \deg r$. Theorem 2.52 generalizes this observation. □

Theorem 2.52 A non-constant polynomial function $G(x,y) \in K[C]$ has only finitely many zeros $P_1, \ldots, P_n \in C_{\mathrm{fin}}$ and a single pole at $\mathcal{O}$. Furthermore, if K is algebraically closed, then $\sum_{i=1}^{n} \mathrm{ord}_{P_i}(G) = -\mathrm{ord}_{\mathcal{O}}(G) = \deg(G)$. ■

2.12.3 The Jacobian

We continue to work with the hyperelliptic curve C of Equation (2.13). We first impose the restriction that K *is algebraically closed* and use the theory of Section 2.10 to define the set $\mathrm{Div}(C)$ of divisors on C, the degree zero part $\mathrm{Div}^0(C)$ of $\mathrm{Div}(C)$, the divisor $\mathrm{Div}(r)$ of a rational function $r \in K(C)$, the set $\mathrm{Prin}(C)$ of principal divisors on C, the Picard group $\mathrm{Pic}(C) = \mathrm{Div}(C)/\mathrm{Prin}(C)$ and the Jacobian $\mathrm{Pic}^0(C) = \mathbb{J}(C) = \mathrm{Div}^0(C)/\mathrm{Prin}(C)$.

Example 2.24 For the rational function $r := (x-h)^m$ of Example 2.23, we have:

$$\mathrm{Div}(r) = \begin{cases} mP + m\tilde{P} - 2m\mathcal{O}, & \text{if } P \in C_{\mathrm{ord}}, \\ 2mP - 2m\mathcal{O}, & \text{if } P \in C_{\mathrm{spl}}. \end{cases}$$

□

The Jacobian $\mathbb{J}(C) = \mathrm{Div}^0(C)/\mathrm{Prin}(C)$ is the set of all cosets of $\mathrm{Prin}(C)$ in $\mathrm{Div}^0(C)$. It is not a good idea to work with cosets (which are equivalence classes). Recall that in the case of $\mathbb{Z}_n = \mathbb{Z}/n\mathbb{Z}$, we represented a coset $a + n\mathbb{Z}$ by the remainder of Euclidean division of a by n. In case of the representation $\mathbb{F}_{p^n}[X] = \mathbb{F}_p[X]/\langle f(X)\rangle$, we took polynomials of smallest degrees as canonical representatives of the cosets of $\langle f(X)\rangle$. In case of $\mathbb{J}(C)$ too, we intend to find such *good* representatives, one from each coset. We now introduce the concept of *reduced divisors* for that purpose.

Definition 2.87 Two divisors $D_1, D_2 \in \mathrm{Div}^0(C)$ (resp. in $\mathrm{Div}(C)$) are said to be *equivalent*, denoted $D_1 \sim D_2$, if $D_1 - D_2 \in \mathrm{Prin}(C)$, or equivalently if $D_1 + \mathrm{Prin}(C) = D_2 + \mathrm{Prin}(C) \in \mathbb{J}(C)$ (resp. $\mathrm{Pic}(C)$). □

Our goal is to associate to every divisor $D \in \mathrm{Div}^0(C)$ some *unique* reduced divisor $D_{\mathrm{red}} \in \mathrm{Div}^0(C)$ with $D \sim D_{\mathrm{red}}$, that is, D_{red} plays the role of the canonical representative of $D + \mathrm{Prin}(C) \in \mathbb{J}(C)$. We start with the following definition.

Definition 2.88 A divisor $\sum_{P\in C_{\mathrm{fin}}} m_P[P] - \left(\sum_{P\in C_{\mathrm{fin}}} m_P\right)[\mathcal{O}] \in \mathrm{Div}^0(C)$ is called *semi-reduced*, if each $m_P \geqslant 0$ and if for $m_P > 0$ we have: $m_{\tilde{P}} = 0$ if P is an ordinary point, and $m_P = 1$ if P is a special point. □

Proposition 2.41 Every divisor $D \in \mathrm{Div}^0(C)$ is equivalent to some semi-reduced divisor D_1.

Proof Let $D = \sum_{P\in C} m_P[P] = \sum_{P\in C_{\mathrm{spl}}} m_P[P] + \sum_{P\in C_1} m_P[P] + \sum_{P\in C_2} m_P[P] - m[\mathcal{O}]$, with $m = \sum_{P\in C_{\mathrm{fin}}} m_P$ and with C_{ord} being the disjoint union of C_1 and C_2, where an ordinary point $P \in C_1$ if and only if its opposite $\tilde{P} \in C_2$ and $m_P \geqslant m_{\tilde{P}}$. Now we can write $D = D_1 + D_2$, where

$$D_1 = \sum_{P\in C_{\mathrm{spl}}} (m_P - 2\lfloor m_P/2\rfloor)[P] + \sum_{P\in C_1} (m_P - m_{\tilde{P}})[P] - m_1[\mathcal{O}]$$

and

$$D_2 = \sum_{P\in C_{\mathrm{spl}}} (2\lfloor m_P/2\rfloor)[P] + \sum_{P\in C_1} m_{\tilde{P}}([P] + [\tilde{P}]) - m_2[\mathcal{O}]$$

with m_1 and m_2 so chosen that $D_1, D_2 \in \mathrm{Div}^0(C)$. By definition, D_1 is semi-reduced, whereas by Example 2.24 $D_2 = \mathrm{Div}(r) \in \mathrm{Prin}(C)$, where

$$r = \left(\prod_{P=(h,k)\in C_{\mathrm{spl}}} (x-h)^{\lfloor m_P/2\rfloor}\right)\left(\prod_{P=(h,k)\in C_1} (x-h)^{m_{\tilde{P}}}\right) \in K(C).$$ ■

Now, we explain how we can represent a semi-reduced divisor by a pair of polynomials $a(x), b(x) \in K[x]$. For that, we need a definition.

Definition 2.89 Let $D_1 = \sum_{P\in C(K)} m_P[P]$ and $D_2 = \sum_{P\in C(K)} n_P[P]$ be two divisors on C (not necessarily in $\mathrm{Div}^0(P)$). The *greatest common divisor* (gcd) of D_1 and D_2 is defined as the *divisor*

$$\begin{aligned}\gcd(D_1, D_2) &:= \sum_{P\in C_{\mathrm{fin}}} \min(m_P, n_P)[P] - \left(\sum_{P\in C_{\mathrm{fin}}} \min(m_P, n_P)\right)[\mathcal{O}] \\ &\in \mathrm{Div}^0(C)\ .\end{aligned}$$

□

Theorem 2.53 Let $D = \sum_{P \in C_{\text{fin}}} m_P[P] - \left(\sum_{P \in C_{\text{fin}}} m_P\right)[\mathcal{O}]$ be a semi-reduced divisor on C. Let $P_i = (h_i, k_i)$, $i = 1, \ldots, n$, be the only finite points P on C such that $m_P > 0$. Let $m_i := m_{P_i}$, $m := \sum_{i=1}^{n} m_i$ and $a(x) := \prod_{i=1}^{n} (x - h_i)^{m_i}$ (so that $\deg_x(a) = m$). Then there exists a unique polynomial $b(x) \in K[x]$ with the following properties:

(1) $\deg_x b < m$,
(2) $b(h_i) = k_i$ for $i = 1, \ldots, n$,
(3) $a(x)$ divides $b(x)^2 + b(x)u(x) - v(x)$, and
(4) $D = \gcd\Big(\text{Div}(a(x)), \text{Div}(b(x) - y)\Big)$.

Conversely, if $a(x), b(x) \in K[x]$ with $\deg_x b < \deg_x a$ and with a dividing $b^2 + bu - v$, then the divisor $\gcd\Big(\text{Div}(a(x)), \text{Div}(b(x) - y)\Big)$ is semi-reduced. ■

We denote the divisor $\gcd\Big(\text{Div}(a(x)), \text{Div}(b(x) - y)\Big)$ by $\text{Div}(a, b)$. The zero divisor has the representation $\text{Div}(1, 0)$.

A representation of the elements of $\mathbb{J}(C)$ by semi-reduced divisors (that is, by pairs of polynomials in $K[x]$) suffers from two disadvantages. First, the representation is not unique, and second, the degrees of the representing polynomials may be quite large. These difficulties are removed if we consider semi-reduced divisors of a special kind.

Definition 2.90 A semi-reduced divisor $D = \sum_{P \in C_{\text{fin}}} m_P[P] - \left(\sum_{P \in C_{\text{fin}}} m_P\right)[\mathcal{O}]$ is called a *reduced divisor*, if $\sum_{P \in C_{\text{fin}}} m_P \leqslant g$, where g is the genus of C. □

The following theorem establishes the desirable properties of a reduced divisor.

Theorem 2.54 For $D \in \text{Div}^0(C)$, there exists a unique reduced divisor D_1 equivalent to D.

Proof We only prove the existence of reduced divisors. For the proof of the uniqueness, one may, for example, see Koblitz [154]. The *norm* of a divisor $D = \sum_{P \in C(K)} m_P[P]$ is defined as the integer $|D| = \sum_{P \in C_{\text{fin}}} |m_P|$.

Let $D = \sum_{P \in C_{\text{fin}}} m_P[P] - \left(\sum_{P \in C_{\text{fin}}} m_P\right)[\mathcal{O}] \in \text{Div}^0(C)$. By Proposition 2.41 there exists a semi-reduced divisor $D' \sim D$. One can easily verify that $|D'| \leqslant |D|$. If we already have $|D'| \leqslant g$, then D' is a desired reduced divisor. So assume otherwise, that is, $|D'| \geqslant g + 1$. We can then choose finite points $P_1, \ldots, P_{g+1}$ on C (not necessarily all distinct) such that $\sum_{i=1}^{g+1}[P_i]$ is a *subsum* of the formal sum D'. Let the semi-reduced divisor $[P_1] + \cdots + [P_{g+1}] - (g+1)[\mathcal{O}]$ be represented as $\text{Div}(a, b)$ with $\deg_x a = g + 1$ and $\deg_x b \leqslant g$. But then $\deg(b(x) - y) = 2g + 1$ and $b(x) - y$ has zeros at $P_1, \ldots, P_{g+1}$ by Theorem 2.53. So by Theorem 2.52 we can write $\text{Div}(b(x) - y) = [P_1] + \cdots + [P_{g+1}] + [Q_1] + \cdots + [Q_g] - (2g + 1)[\mathcal{O}]$ for some finite points $Q_1, \ldots, Q_g$ on C. Now $D'' := D' - \text{Div}(b(x) - y) \in \text{Div}^0(C)$ satisfies $D'' \sim D'$ and $|D''| < |D'|$. We apply Proposition 2.41 again to get a semi-reduced divisor $D''' \sim D''$ with $|D'''| \leqslant |D''|$. Thus starting from the semi-reduced divisor D' we produce another semi-reduced divisor D''' such that $D''' \sim D' \sim D$ and $|D'''| < |D'|$. We continue the process a finite number of times, until we get an equivalent semi-reduced divisor D_1 of norm $\leqslant g$. This is a desired reduced divisor. ■

From the viewpoint of cryptography, the field K should be a finite field which is *never* algebraically closed. So we must remove the restriction $\bar{K} = K$. Since C

is naturally defined over $\bar{K}$ as well, we start with the Jacobian $\mathbb{J}_{\bar{K}}(C)$ and define a particular subgroup of $\mathbb{J}_{\bar{K}}(C)$ to be the Jacobian $\mathbb{J}_K(C)$ of C over K.

Definition 2.91 Let $\varphi \in \mathrm{Aut}_K(\bar{K})$ be a K-automorphism of $\bar{K}$. For a point $P = (h, k) \in C_{\mathrm{fin}}(\bar{K})$, the point $\varphi(P) := (\varphi(h), \varphi(k))$ is also in $C_{\mathrm{fin}}(\bar{K})$. For a divisor $D = \sum_{P \in C(\bar{K})} m_P[P] \in \mathrm{Div}_{\bar{K}}(C)$, we define $\varphi(D) := \sum_{P \in C(\bar{K})} m_P[\varphi(P)]$. D is said to be *defined over* K if $\varphi(D) = D$ for all $\varphi \in \mathrm{Aut}_K(\bar{K})$. The subset of $\mathbb{J}_{\bar{K}}(C)$ consisting of divisor classes that have representative divisors defined over K is a subgroup (denoted by $\mathbb{J}_K(C)$) of $\mathbb{J}_{\bar{K}}(C)$ and is called the *Jacobian of C over K*. □

Every element of $\mathbb{J}_K(C)$ can be represented uniquely as a reduced divisor $\mathrm{Div}(a, b)$ for polynomials $a(x), b(x) \in K[x]$ with $\deg_x a \leqslant g$ and $\deg_x b < \deg_x a$. $\mathbb{J}_K(C)$ is, therefore, a *finite* Abelian group. For suitably chosen hyperelliptic curves, these groups can be used to build cryptographic protocols.

EXERCISE SET 2.12

In this exercise set, we let C denote a hyperelliptic curve of genus g defined by Equation (2.13) over a field K (not necessarily algebraically closed).

2.118 **(a)** Show that the curve

$$C_1 : Y^2 = X^5 + X + 1$$

defined over $\mathbb{F}_7$ is not smooth and so *not* a hyperelliptic curve. Find a point where C_1 is not smooth.

(b) Show that the curve

$$C_2 : Y^2 = X^5 + X + 2$$

defined over $\mathbb{F}_7$ is smooth, that is, a hyperelliptic curve of genus 2. Find out all the $\mathbb{F}_7$-rational points on C_2. (There are ten of them.)

2.119 Represent $\mathbb{F}_8$ as $\mathbb{F}_2[\xi]$, where ξ is a root of the irreducible polynomial $X^3 + X + 1 \in \mathbb{F}_2[X]$.

(a) Show that the curve

$$C_3 : Y^2 + XY = X^5 + X + 1$$

defined over $\mathbb{F}_8$ is not smooth and so *not* a hyperelliptic curve. Find a point where C_3 is not smooth.

(b) Show that the curve

$$C_4 : Y^2 + XY = X^5 + X + \xi$$

defined over $\mathbb{F}_8$ is smooth, that is, a hyperelliptic curve of genus 2. Find out all the $\mathbb{F}_8$-rational points on C_4. (There are eight of them.)

2.120 Let $P = (h, k) \in C_{\mathrm{fin}}$. Prove the following assertions:

(a) The only points on C with X-coordinate equal to h are P and $\tilde{P}$.

(b) $\tilde{\tilde{P}} = P$.
(c) P is a special point if and only if $u^2(h) + 4v(h) = 0$.
(d) If $\mathrm{char}\, K \neq 2$, then C has at most $2g + 1$ special points, whereas if $\mathrm{char}\, K = 2$, then C has at most g special points.

2.121 Prove Lemmas 2.9 and 2.10.

2.122 Let $P = (h, k) \in C_{\mathrm{fin}}$ and $0 \neq G \in K[C]$.
(a) Show that $G(P) = 0$ if and only if $\hat{G}(\tilde{P}) = 0$.
(b) Let $G(P) = \hat{G}(P) = 0$. Show that either P is a special point of C or h is a common root of u and v.
(c) Show that $\mathrm{ord}_P(G) = \mathrm{ord}_{\tilde{P}}(\hat{G})$ and that $\mathrm{ord}_{\mathcal{O}}(G) = \mathrm{ord}_{\mathcal{O}}(\hat{G})$.

2.123 Prove Theorem 2.52. **[H]**

2.124 A *line* on C is a polynomial function of the form $l(x, y) = ax + by + c \in \bar{K}[C]$ with $a, b, c \in \bar{K}$, a and b not both 0.
(a) Let $D = \mathrm{Div}(l)$ be the divisor of a line l. Show that the norm $|D|$ is either 2 or $2g + 1$.
(b) Let $h \in \bar{K}^*$. Determine $\mathrm{Div}(x - h)$.
(c) Determine $\mathrm{Div}(y)$.

2.125 Let E be an elliptic curve (that is, a hyperelliptic curve of genus 1) defined over K.
(a) Show that any divisor $D \in \mathrm{Div}^0_{\bar{K}}(E)$ can be written as $D = [P] - [\mathcal{O}] + \mathrm{Div}(r)$ for some *unique* point $P \in E(\bar{K})$ and for some rational function $r \in \bar{K}(E)$. This rational function r is unique up to multiplication by elements of $\bar{K}^*$.
(b) Show that the map $\sigma : \mathbb{J}_{\bar{K}}(E) \to E(\bar{K})$ that maps the residue class of $D \in \mathrm{Div}^0_{\bar{K}}(E)$ to the point $P \in E(\bar{K})$ satisfying $D = [P] - [\mathcal{O}] + \mathrm{Div}(r)$ for some $r \in \bar{K}(E)$, is a bijection.
(c) Let $P, Q \in E(\bar{K})$, not both $\mathcal{O}$. Show that there is a line l with $\mathrm{Div}(l) = [P] + [Q] + [R] - 3[\mathcal{O}]$, where $R = -(P + Q)$.
(d) Let $\varphi := \sigma^{-1}$, where σ is defined in Part (b). Show that for $P, Q \in E(\bar{K})$ one has $\varphi(P + Q) = \varphi(P) + \varphi(Q)$. (This, in particular, proves Theorem 2.46 and that σ is a group isomorphism.)
(e) Let $D = \sum_{P \in E(\bar{K})} m_P[P] \in \mathrm{Div}_{\bar{K}}(E)$. Show that D is a principal divisor if and only if $\sum_{P \in E(\bar{K})} m_P = 0$ (integer sum) and $\sum_{P \in E(\bar{K})} m_P P = \mathcal{O}$ (sum in $E(\bar{K})$).

** 2.13 Number Fields

In this section, we develop the theory of number fields and rings. Our aim is to make accessible to the readers the working of the cryptanalytic algorithms based on number field sieves.

2.13.1 Some Commutative Algebra

Commutative algebra is the study of commutative rings with identity (rings by *our* definition). Modern number theory and geometry are based on results from this area of mathematics. Here we give a brief sketch of some commutative algebra tools that we need for developing the theory of number fields.

Ideal arithmetic

We start with some basic operations on ideals (cf. Example 2.7, Definition 2.23).

Definition 2.92 Let A be a ring and let $\mathfrak{a}_i$, $i \in I$, be a family (not necessarily finite) of ideals in A.

The set-theoretic *intersection* $\bigcap_{i\in I} \mathfrak{a}_i$ is evidently an ideal in A.

The *sum* of the family $\mathfrak{a}_i$ is the ideal

$$\sum_{i\in I} \mathfrak{a}_i := \left\{ \sum_{i\in I} x_i \mid x_i \in \mathfrak{a}_i \text{ and } x_i = 0 \text{ except for finitely many } i \in I \right\}.$$

Two ideals $\mathfrak{a}$ and $\mathfrak{b}$ of A are said to be *relatively prime* or *coprime*, if $\mathfrak{a} + \mathfrak{b} = A$, or equivalently if there exist $a \in \mathfrak{a}$ and $b \in \mathfrak{b}$ with $a + b = 1$.

If $I = \{1, 2, \ldots, n\}$ is *finite*, the *product* $\mathfrak{a}_1\mathfrak{a}_2 \cdots \mathfrak{a}_n$ is the ideal generated by all elements of the form $x_1x_2 \cdots x_n$ with $x_i \in \mathfrak{a}_i$ for all $i = 1, \ldots, n$. We have:

$$\mathfrak{a}_1\mathfrak{a}_2 \cdots \mathfrak{a}_n = \left\{ \sum_{j=1}^{r} x_{j,1}x_{j,2} \cdots x_{j,n} \mid r \in \mathbb{N}_0, x_{j,i} \in \mathfrak{a}_i \right\}.$$

If $\mathfrak{a}_1 = \mathfrak{a}_2 = \cdots = \mathfrak{a}_n = \mathfrak{a}$, the product $\mathfrak{a}_1\mathfrak{a}_2 \cdots \mathfrak{a}_n$ is denoted as $\mathfrak{a}^n$. The empty product of ideals is conventionally taken to be the unit ideal A. If $\mathfrak{a}$ is the principal ideal $\langle a \rangle$, then $\mathfrak{a}^n = \langle a^n \rangle$. □

One can readily check that the operations intersection, sum and product on ideals in a ring are associative and commutative.

Commutative algebra extensively uses the theory of prime and maximal ideals (Definition 2.19, Proposition 2.9, Corollary 2.2 and Exercise 2.23). The set of all prime ideals in A is called the *(prime) spectrum* of A and is denoted by $\operatorname{Spec} A$. The set of all maximal ideals of A is called the *maximal spectrum* of A and denoted by $\operatorname{Spm} A$. We have $\operatorname{Spm} A \subseteq \operatorname{Spec} A$. These two sets play an extremely useful role for the study of the ring A. If A is non-zero, both these sets are non-empty.

Localization

The concept of formation of fractions of integers to give the rationals can be applied in a more general setting. Instead of having any non-zero element in the denominator of a fraction we may allow only elements from a specific subset. All we require to make the collection of fractions a ring is that the allowed denominators should be closed under multiplication.

Definition 2.93 Let A be a ring. A non-empty subset S of A is called *multiplicatively closed* or simply *multiplicative*, if $1 \in S$ and for any $s, t \in S$ we have $st \in S$. □

Example 2.25

(1) For a non-zero ring A, the subset $A \setminus \{0\}$ is multiplicatively closed, if and only if A is an integral domain. For a general non-zero ring A, the set of all elements $a \in A$ such that a is not a zero-divisor is a multiplicative subset of A.

(2) Let A be a ring and $\mathfrak{a}$ a proper ideal of A. The set $A \setminus \mathfrak{a}$ is multiplicatively closed, if and only if $\mathfrak{a}$ is a prime ideal of A.

(3) For a ring A and an element $f \in A$, the set $\{1, f, f^2, f^3, \ldots\} \subseteq A$ is multiplicatively closed. □

Let A be a ring and S a multiplicative subset of A. We define a relation $\sim$ on $A \times S$ as: $(a, s) \sim (b, t)$ if and only if $u(at - bs) = 0$ for some $u \in S$. (If A is an integral domain, one may take $u = 1$ in the definition of $\sim$.) It is easy to check that $\sim$ is an equivalence relation on $A \times S$. The set of equivalence classes of $A \times S$ under $\sim$ is denoted by $S^{-1}A$, whereas the equivalence class of $(a, s) \in A \times S$ is denoted as a/s. For $a/s, b/t \in S^{-1}A$, define $(a/s) + (b/t) := (at + bs)/(st)$ and $(a/s)(b/t) := (ab)/(st)$. It is easy to check that these operations are well-defined and make $S^{-1}A$ a ring with identity $1/1$, in which each $s/1$, $s \in S$, is invertible. There is a canonical ring homomorphism $\imath : A \to S^{-1}A$ taking $a \mapsto a/1$. In general, $\imath$ is *not* injective. However, if A is an integral domain and $0 \notin S$, then the injectivity of $\imath$ can be proved easily and we say that the ring A is canonically embedded in the ring $S^{-1}A$.

Definition 2.94 Let A be a ring and S a multiplicative subset of A. The ring $S^{-1}A$ constructed as above is called the *localization of A away from S* or the *ring of fractions* of A with respect to S. □

Example 2.26

(1) Let A be an integral domain and let $S = A \setminus \{0\}$. Then $S^{-1}A$ is called the *quotient field* or the *field of fractions* of A and is denoted as $\mathrm{Q}(A)$. If A is already a field, then $\mathrm{Q}(A) \cong A$. Other examples include $\mathrm{Q}(\mathbb{Z}) = \mathbb{Q}$ and $\mathrm{Q}(K[X]) = K(X)$, K a field, where $K(X)$ denotes the field of rational functions over K in one indeterminate X.

More generally, if A is any ring and S is the set of all non-zero-divisors of A, then $S^{-1}A$ is called the *total quotient ring* of A and is again denoted by $\mathrm{Q}(A)$. It is, in general, not a field. If A is an integral domain, then $S = A \setminus \{0\}$ and the usage of $\mathrm{Q}(A)$ remains consistent.

(2) Let A be a ring, $\mathfrak{p}$ a prime ideal of A and $S := A \setminus \mathfrak{p}$. Then $S^{-1}A$ is called the *localization of A at $\mathfrak{p}$* and is usually denoted by $A_{\mathfrak{p}}$.

(3) Let A be a ring, $f \in A$ and $S = \{1, f, f^2, f^3, \ldots\}$. In this case, $S^{-1}A$ is conventionally denoted by A_f. □

Integral dependence

The concept of integral dependence generalizes the notion of *integers*. Recall that for a field extension $K \subseteq L$, an element $\alpha \in L$ is called algebraic over K, if α is a root

of a non-zero polynomial $f \in K[X]$. Since K is a field, the polynomial f can be divided by its leading coefficient, giving a *monic* polynomial in $K[X]$ of which α is a root. However, if K is not a field, division by the leading coefficient is not always permissible. So we require the minimal polynomial to be monic in order to define a special class of objects.

Definition 2.95 Let $A \subseteq B$ be an extension of rings. An element $\alpha \in B$ is said to be *integral over* A, if α satisfies[15] (that is, is a root of) a *monic* (and hence non-zero) polynomial $f \in A[X]$. An equation of the form $f(\alpha) = 0$, $f \in A[X]$ monic, is called an equation of *integral dependence* of α over A. □

Example 2.27

(1) If both A and B are fields, the concepts of *integral* and *algebraic* elements are the same. (See the argument preceding Definition 2.95.)

(2) Take $A := \mathbb{Z}$ and $B := \mathbb{Q}$ and let $a/b \in \mathbb{Q}$, $\gcd(a, b) = 1$, be integral over $\mathbb{Z}$. Let $(a/b)^n + \alpha_{n-1}(a/b)^{n-1} + \cdots + \alpha_1(a/b) + \alpha_0$, $\alpha_i \in \mathbb{Z}$, be an equation of integral dependence of a/b over $\mathbb{Z}$. Multiplication by b^n gives $a^n = -b(\alpha_{n-1}a^{n-1} + \cdots + \alpha_1 ab^{n-2} + \alpha_0 b^{n-1})$, that is, $b|a^n$. Since $\gcd(a, b) = 1$, this forces $b = \pm 1$, that is, $a/b \in \mathbb{Z}$. This is, in general, true for any UFD A and its field of fractions $B = \mathrm{Q}(A)$ (See Exercise 2.131).

(3) Every element $\alpha \in A$ is integral over A, since it satisfies the monic polynomial $X - \alpha \in A[X]$. □

Now let $A \subseteq B$ be an extension of rings and let C consist of all the elements of B that are integral over A. Clearly, $A \subseteq C \subseteq B$. It turns out that C is again a ring. This result is not at all immediate from the definition of integral elements. We prove this by using the following lemma which generalizes Theorem 2.33.

Lemma 2.11 For a ring extension $A \subseteq B$ and for $\alpha \in B$, the following conditions are equivalent:

(a) α is integral over A.
(b) $A[\alpha]$ is a finitely generated A-module.
(c) $A[\alpha] \subseteq C$ for some subring C of B with C being a finitely generated A-module.

Proof [(a)⇒(b)] Let $\alpha^n + a_{n-1}\alpha^{n-1} + \cdots + a_1\alpha + a_0 = 0$, $a_i \in A$, be an equation of integral dependence of α over A. $A[\alpha] = \{f(\alpha) \mid f(X) \in A[X]\}$ is generated as an A-module by $1, \alpha, \alpha^2, \ldots$. In order to show that only the elements $1, \alpha, \ldots, \alpha^{n-1}$ generate $A[\alpha]$ as an A-module, it is sufficient to show that each α^k, $k \in \mathbb{N}_0$, is an A-linear combination of $1, \alpha, \ldots, \alpha^{n-1}$. We proceed by induction on k. The assertion certainly holds for $k = 0, \ldots, n-1$, whereas for $k \geqslant n$ we write $\alpha^k = -(a_{n-1}\alpha^{k-1} + \cdots + a_1\alpha^{k-n+1} + a_0\alpha^{k-n})$, whence induction completes the proof.
[(b)⇒(c)] Take $C := A[\alpha]$.
[(c)⇒(a)] Let $\gamma_1, \ldots, \gamma_n \in C$ generate C as an A-module. Since $A[\alpha] \subseteq C$ and, in particular, $\alpha \in C$, for all $i = 1, \ldots, n$ we can write $\alpha\gamma_i = \sum_{j=1}^{n} a_{ij}\gamma_j$ for some $a_{ij} \in A$. Let $\mathfrak{A}$ denote the matrix $(\alpha\delta_{ij} - a_{ij})_{1 \leqslant i,j \leqslant n}$, where δ_{ij} is the Kronecker delta. Then $\mathfrak{A}(\gamma_1, \ldots, \gamma_n)^{\mathrm{t}} = (0, \ldots, 0)^{\mathrm{t}}$. Multiplication (on the left) by the adjoint of

[15] Strictly speaking, α being a root of $f(X)$ is equivalent to α satisfying the polynomial *equation* $f(\alpha) = 0$. Often the term *equation* is dropped in this context—a harmless colloquial contraction.

$\mathfrak{A}$ shows that $(\det \mathfrak{A})\gamma_i = 0$ for all $i = 1, \ldots, n$. Since $1 \in C$, we have $1 = \sum_{i=1}^{n} b_i\gamma_i$ for some $b_i \in A$, so that $(\det \mathfrak{A}) \cdot 1 = 0$, that is, $\det \mathfrak{A} = 0$. But $\det \mathfrak{A}$ is a monic polynomial in α of degree n and with coefficients from A. ■

Proposition 2.42 For an extension $A \subseteq B$ of rings, the set

$$C := \{\alpha \in B \mid \alpha \text{ is integral over } A\}$$

is a subring of B containing A.

Proof Clearly, $A \subseteq C \subseteq B$ as sets. To show that C is a ring let $\alpha, \beta \in C$. By Condition (b) of Lemma 2.11, $A[\alpha]$ is a finitely generated A-module. Now β, being integral over A, is also integral over $A[\alpha]$; so again by Lemma 2.11(b), $A[\alpha][\beta]$ is a finitely generated $A[\alpha]$-module. It is then easy to check that $A[\alpha, \beta] = A[\alpha][\beta]$ is a finitely generated A-module. Since $\alpha \pm \beta$ and $\alpha\beta$ are in $A[\alpha, \beta]$, by Lemma 2.11(c), these elements are integral over A, that is, belong to C. Thus C is a ring. ■

Definition 2.96 The ring C of Proposition 2.42 is called the *integral closure* of A in B. A is called *integrally closed* in B, if $C = A$. On the other hand, if $C = B$, we say that B is an *integral extension* of A or that B is *integral* over A.

An integral domain A is called *integrally closed* (without specific mention of the ring in which it is so), if A is integrally closed in its quotient field $\mathrm{Q}(A)$. An integrally closed integral domain is called a *normal domain* (*ND*). □

Example 2.28

(1) $\mathbb{Z}$ (or more generally any UFD) is a normal domain.

(2) $\mathbb{Z}$ is not integrally closed in $\mathbb{R}$ or $\mathbb{C}$, since, for example, $\sqrt{2} \in \mathbb{R} \setminus \mathbb{Z}$ is integral over $\mathbb{Z}$. The integral closure of $\mathbb{Z}$ in $\mathbb{C}$ is denoted by $\mathbb{A}$. Elements of $\mathbb{A}$ are called *algebraic integers* (See Exercise 2.60). □

Noetherian rings

Recall that a PID is a ring (integral domain) in which every ideal is principal, that is, generated by a single element. We now want to be a bit more general and demand every ideal to be finitely generated. If a ring meets our demand, we call it a Noetherian ring. These rings are named after Emmy Noether (1882–1935) who was one of the most celebrated lady mathematicians of all ages and whose work on Noetherian rings has been very fundamental and deep in the branch of algebra. Emmy's father Max Noether (1844–1921) was also an eminent mathematician.

Definition 2.97 Let A be a ring and let $\mathfrak{a}_1 \subseteq \mathfrak{a}_2 \subseteq \mathfrak{a}_3 \subseteq \cdots$ be an *ascending chain* of ideals of A. This chain is called *stationary*, if there is an $n \in \mathbb{N}$ such that $\mathfrak{a}_n = \mathfrak{a}_{n+1} = \mathfrak{a}_{n+2} = \cdots$. The ring A is said to satisfy the *ascending chain condition* or the *ACC*, if every ascending chain of ideals in A is stationary, or in other words, if there does not exist any infinite strictly ascending chain $\mathfrak{a}_1 \subsetneqq \mathfrak{a}_2 \subsetneqq \mathfrak{a}_3 \subsetneqq \cdots$ of ideals in A. □

Proposition 2.43 For a ring A, the following conditions are equivalent:

(a) Every ideal of A is finitely generated.

(b) A satisfies the ascending chain condition.

(c) Every non-empty set of ideals of A contains a maximal element.

Proof [(a)⇒(b)] Let $\mathfrak{a}_1 \subseteq \mathfrak{a}_2 \subseteq \cdots$ be an ascending chain of ideals of A. Consider the ideal $\mathfrak{a} := \bigcup_{n\in\mathbb{N}} \mathfrak{a}_n$ which is finitely generated by hypothesis. Let $a_1, \ldots, a_r$ be a set of generators of $\mathfrak{a}$. Each $a_i \in \mathfrak{a}$, that is, there exists $m_i \in \mathbb{N}$ such that $a_i \in \mathfrak{a}_{m_i}$ and hence $a_i \in \mathfrak{a}_n$ for every $n \geqslant m_i$. Take $m := \max(m_1, \ldots, m_r)$. For every $n \geqslant m$, we have $\mathfrak{a} \subseteq \mathfrak{a}_n$, that is, $\mathfrak{a} = \mathfrak{a}_n$.
[(b)⇒(c)] Let S be a non-empty set of ideals of A. Order S by inclusion. The ACC implies that every chain in S has an upper bound in S. By Zorn's lemma, S has a maximal element.
[(c)⇒(a)] Let $\mathfrak{a}$ be an ideal of A. Consider the set S of all finitely generated ideals of A contained in $\mathfrak{a}$. S is non-empty, since it contains the zero ideal. By condition (c), S has a maximal element, say, $\mathfrak{b}$. If $\mathfrak{b} \subsetneqq \mathfrak{a}$, take $a \in \mathfrak{a} \setminus \mathfrak{b}$. Then $\mathfrak{b} + \langle a\rangle$ is finitely generated (since $\mathfrak{b}$ is so), properly contains $\mathfrak{b}$ and is contained in $\mathfrak{a}$. This contradicts the maximality of $\mathfrak{b}$ in S. Thus we must have $\mathfrak{a} = \mathfrak{b}$, that is, $\mathfrak{a}$ is finitely generated. ■

Definition 2.98 A ring A is called *Noetherian*, if A satisfies (one and hence all of) the equivalent conditions of Proposition 2.43. □

Example 2.29

(1) All PIDs are Noetherian, since principal ideals are obviously finitely generated. In particular, $\mathbb{Z}$ and $K[X]$ (K a field) are Noetherian.

(2) If A is Noetherian and $\mathfrak{a}$ an ideal of A, then $A/\mathfrak{a}$ is Noetherian, since the ideals of $A/\mathfrak{a}$ are in one-to-one inclusion-preserving correspondence with the ideals of A containing $\mathfrak{a}$ and hence satisfy the ACC.

(3) Let A be a Noetherian ring and S a multiplicative subset of A. Then the localization $B := S^{-1}A$ is also Noetherian. To prove this fact let $\mathfrak{b}$ be an ideal in B. One can show that $\mathfrak{b} = S^{-1}\mathfrak{a}$ for some ideal $\mathfrak{a}$ of A. Since A is Noetherian, $\mathfrak{a}$ is finitely generated, say, $\mathfrak{a} = \langle a_1, \ldots, a_r\rangle$. It is now (almost) obvious that $\mathfrak{b}$ is generated by $a_1/1, \ldots, a_r/1$. A particular case: If A is Noetherian and $\mathfrak{p}$ a prime ideal of A, then the localization $A_{\mathfrak{p}}$ is also Noetherian.

(4) The ring $A := \mathbb{Z}[X_1, X_2, X_3, \ldots]$ of polynomials with infinitely many indeterminates $X_1, X_2, X_3, \ldots$ is not Noetherian. This is because the ideal

$$\langle X_1, X_2, X_3, \ldots\rangle = AX_1 + AX_2 + AX_3 + \cdots$$

is not finitely generated, or alternatively because we have the infinite strictly ascending chain of ideals: $\langle X_1\rangle \subsetneqq \langle X_1, X_2\rangle \subsetneqq \langle X_1, X_2, X_3\rangle \subsetneqq \cdots$, or because the set $S := \{\langle X_1\rangle, \langle X_1, X_2\rangle, \langle X_1, X_2, X_3\rangle, \ldots\}$ of ideals in A does not contain a maximal element. □

We have seen that if A is a PID, the polynomial ring $A[X]$ need not be a PID. However, the property of being Noetherian is preserved during the passage from A to $A[X]$ (Theorem 2.8).

Dedekind domains

A class of rings proves to be vital in the study of number fields:

Definition 2.99 An integral domain A is called a *Dedekind domain*, if it satisfies *all* of the following three conditions:

(1) A is Noetherian.

(2) Every *non-zero* prime ideal of A is maximal.

(3) A is integrally closed (in its quotient field $K := \mathrm{Q}(A)$). □

2.13.2 Number Fields and Rings

After much ado we are finally in a position to define the basic objects of study in this section.

Definition 2.100 A *number field* K is defined to be a finite (and hence algebraic) extension of the field $\mathbb{Q}$ of rational numbers. Clearly, $\mathbb{Q} \subseteq K \subseteq \mathbb{C}$. The extension degree $[K : \mathbb{Q}]$ is called the *degree* of the number field K and is finite by definition. □

Note that there exist considerable controversies among mathematicians in accepting this definition of number fields. Some insist that any field K satisfying $\mathbb{Q} \subseteq K \subseteq \mathbb{C}$ should be called a number field. Some others restrict the definition by demanding that one must have K *algebraic* over $\mathbb{Q}$; however, fields K with infinite extension degree $[K : \mathbb{Q}]$ are allowed. We restrict the definition further by imposing the condition that $[K : \mathbb{Q}]$ has to be finite. Our restricted definition is seemingly the most widely accepted one. In this book, we study only the number fields of Definition 2.100 and accepting this definition would at the minimum save us from writing huge expressions like "(algebraic) number fields of finite extension degree over $\mathbb{Q}$" to denote *number fields*.

For number fields, the notion of integral closure leads to the following definition.

Definition 2.101 A number field K contains $\mathbb{Q}$ and hence $\mathbb{Z}$. The integral closure of $\mathbb{Z}$ in K is called the *ring of integers* of K and is denoted by $\mathfrak{O}_K$. ($\mathfrak{O}$ is the Gothic O.) Clearly, $\mathbb{Z} \subseteq \mathfrak{O}_K \subseteq K$ and $\mathfrak{O}_K$ is an integral domain. We also have $\mathfrak{O}_K = \mathbb{A} \cap K$, where $\mathbb{A}$ is the subset of $\mathbb{C}$ comprising all algebraic integers. A *number ring* is a ring which is (isomorphic to) the ring of integers of a number field. □

By Example 2.27(2), the ring of integers of the number field $\mathbb{Q}$ is $\mathbb{Z}$, that is, $\mathfrak{O}_{\mathbb{Q}} = \mathbb{Z}$. It is, therefore, customary to call the elements of $\mathbb{Z}$ *rational integers*. Since $\mathbb{Z}$ is naturally embedded in $\mathfrak{O}_K$ for any number field K, it is important to notice the distinction between the integers of K (that is, the elements of $\mathfrak{O}_K$) and the rational integers of K (that is, the images of the canonical inclusion $\mathbb{Z} \hookrightarrow K$).

Some simple properties of number rings are listed below.

Proposition 2.44 For a number field K, we have:

(1) $\mathfrak{O}_K \cap \mathbb{Q} = \mathbb{Z}$.

(2) For $\alpha \in K$, there exists a rational integer $a \in \mathbb{Z}$ such that $a\alpha \in \mathfrak{O}_K$. In particular, the quotient field of $\mathfrak{O}_K$ is K.

(3) $\mathfrak{O}_K$ is integrally closed in $K = \mathrm{Q}(\mathfrak{O}_K)$, that is, $\mathfrak{O}_K$ is a normal domain.

Proof (1) follows immediately from Example 2.27(2), (2) follows from Exercise 2.60, and (3) follows from Exercise 2.126(b). ■

Let K be a number field of degree d. By Corollary 2.13, K is a simple extension of $\mathbb{Q}$, that is, there exists an element $\alpha \in K$ with a minimal polynomial $f(X)$ over $\mathbb{Q}$ such that $\deg f(X) = d = [K : \mathbb{Q}]$ and $K = \mathbb{Q}(\alpha) = \mathbb{Q}[\alpha] \cong \mathbb{Q}[X]/\langle f(X)\rangle$. The field K is a $\mathbb{Q}$-vector space of dimension d with basis $1, \alpha, \ldots, \alpha^{d-1}$. There exists a non-zero integer a such that $\beta := a\alpha \in K$ is an algebraic integer and we continue to have $K = \mathbb{Q}(\beta)$. Thus, without loss of generality, we may take α to be an algebraic integer. In this case, the $\mathbb{Q}$-basis $1, \alpha, \ldots, \alpha^{d-1}$ of K consists only of algebraic integers.

Conversely, let $f(X) \in \mathbb{Q}[X]$ be an irreducible polynomial of degree $d \geqslant 1$. The field $K := \mathbb{Q}[X]/\langle f(X)\rangle$ is a number field of degree d and the elements of K can be represented by polynomials with rational coefficients and of degrees $< d$. Arithmetic in K is carried out as the polynomial arithmetic of $\mathbb{Q}[X]$ followed by reduction modulo the defining irreducible polynomial $f(X)$. This gives us an algebraic representation of K independent of any element of K. Now, K can also be viewed as a subfield of $\mathbb{C}$ and the elements of K can be represented as complex numbers.[16] A representation $K' \subseteq \mathbb{C}$ with a field isomorphism $\sigma : K = \mathbb{Q}[X]/\langle f(X)\rangle \to K'$ is called a *complex embedding* of K in $\mathbb{C}$.[17] Such a representation is not unique as Proposition 2.45 demonstrates.

Proposition 2.45 A number field K of degree $d \geqslant 1$ has exactly d distinct complex embeddings.

Proof As above we take $K := \mathbb{Q}[X]/\langle f(X)\rangle$ for some irreducible polynomial $f(X) \in \mathbb{Q}[X]$ of degree d. Since $\mathbb{Q}$ is a perfect field (See Exercise 2.76), the d roots $\alpha_1, \ldots, \alpha_d \in \mathbb{C}$ of $f(X)$ are all distinct. For each $i = 1, \ldots, d$, the map sending $X + \langle f(X)\rangle \mapsto \alpha_i$ clearly extends to a field isomorphism $\sigma_i : \mathbb{Q}[X]/\langle f(X)\rangle \to \mathbb{Q}(\alpha_i)$. Thus we get d distinct complex embeddings $\mathbb{Q}(\alpha_i) \subseteq \mathbb{C}$ of K in $\mathbb{C}$. Now let K' be a subfield of $\mathbb{C}$, such that $\sigma : \mathbb{Q}[X]/\langle f(X)\rangle \to K'$ is a $\mathbb{Q}$-isomorphism. Let $\alpha := \sigma(X + \langle f(X)\rangle)$. Then $0 = \sigma(0) = \sigma(f(X + \langle f(X)\rangle)) = f(\sigma(X + \langle f(X)\rangle)) = f(\alpha)$. Thus α is a root of f, that is, $\alpha = \alpha_i$ for some $i \in \{1, \ldots, d\}$. Since K' is a field containing $\mathbb{Q}$ and α_i and having $[K' : \mathbb{Q}] = [K : \mathbb{Q}] = d$, it follows that $K' = \mathbb{Q}(\alpha_i)$ and $\sigma = \sigma_i$. ■

This proposition says that the conjugates $\alpha_1, \ldots, \alpha_d$ are *algebraically indistinguishable*. For example, $X^2 + 1$ has two roots $\pm\mathrm{i}$, where $\mathrm{i} = \sqrt{-1}$. But it makes little sense to talk about the *positive* and the *negative* square roots of -1? They are algebraically indistinguishable and if one calls one of these i, the other one becomes $-\mathrm{i}$.[18] However, if a representation of $\mathbb{C}$ is given, we can distinguish between $+\sqrt{-5}$ and $-\sqrt{-5}$ by associating these quantities with the elements $\mathrm{i}\sqrt{5}$ and $-\mathrm{i}\sqrt{5}$ respectively, where $\sqrt{5}$ is the *positive* real square root of 5 and where $\mathrm{i} = \sqrt{-1}$ is the imaginary *unit* available from the given representation of $\mathbb{C}$.

[16] A complex number $z := a + \mathrm{i}b \in \mathbb{C}$ has a representation by a pair (a, b) of real numbers. Here, $\mathrm{i} := \sqrt{-1}$ plays the role of $X + \langle X^2 + 1\rangle$ in $\mathbb{R}[X]/\langle X^2 + 1\rangle$. Finally, every real number has a decimal (or binary or hexadecimal or ...) representation.

[17] The field $\mathbb{Q}$ is canonically embedded in K. It is evident that the embedding $\sigma : K \to K'$ fixes $\mathbb{Q}$ element-wise.

[18] In a number theory seminar in 1996, Hendrik W. Lenstra, Jr. commented:
Suppose the Martians defined the complex numbers by adjoining a root of -1 they called j. And when the Earth and Martians start talking, they have to translate i to be either j or $-j$. So we take i to j, because I think that's what the scientists will decide. ⋯ But it was later discovered that most Martians are left handed, so the philosophers decide it's better to send i to $-j$ instead.

It is also quite customary to start with $K = \mathbb{Q}(\alpha)$ for some algebraic $\alpha \in \mathbb{C}$ and seek for the *complex embeddings* of K in $\mathbb{C}$. One then considers the minimal polynomial $f(X)$ of α (over $\mathbb{Q}$) and proceeds as in the proof of Proposition 2.45 but now defining the map $\sigma_i : \mathbb{Q}(\alpha) \to \mathbb{Q}(\alpha_i)$ as the unique field isomorphism that fixes $\mathbb{Q}$ and takes $\alpha \mapsto \alpha_i$. If we take $\alpha = \alpha_1$, then σ_1 is the identity map, whereas $\sigma_2, \ldots, \sigma_d$ are non-identity field isomorphisms.

The moral of this story is that whether one wants to view the number field K as $\mathbb{Q}[X]/\langle f(X)\rangle$ or as $\mathbb{Q}(\alpha_i)$ for any $i \in \{1, \ldots, d\}$ is one's personal choice. In any case, one will be dealing with the same mathematical object and as long as representation issues are not brought into the scene, all these definitions of a number field are absolutely equivalent.

The embeddings $\mathbb{Q}(\alpha_i)$ need not be all distinct as sets. For example, the two embeddings $\mathbb{Q}(\mathrm{i})$ and $\mathbb{Q}(-\mathrm{i})$ of $\mathbb{Q}[X]/\langle X^2+1\rangle$ are identical as sets. But the maps $x \mapsto \mathrm{i}$ and $x \mapsto -\mathrm{i}$ are distinct (where $x := X + \langle X^2+1\rangle$). Thus while specifying a complex embedding of a number field K, it is necessary to mention not only the subfield K' of $\mathbb{C}$ isomorphic to K, but also the explicit field isomorphism $K \to K'$.

Definition 2.102 Let K be a number field of degree d defined by an irreducible polynomial $f(X) \in \mathbb{Q}[X]$ or by any root of $f(X)$. Let r_1 be the number of real roots and $2r_2$ the number of non-real roots of f. (Note that the non-real roots of a real polynomial occur in (complex) conjugates.) By the fundamental theorem of algebra, we have $d = r_1 + 2r_2$. For any real root α of f, the complex embedding $\mathbb{Q}(\alpha)$ of K is completely contained in $\mathbb{R}$ and hence is often called a *real embedding* of K. On the other hand, for a non-real root β of f the complex embedding $\mathbb{Q}(\beta)$ of K is called a *non-real* or a *properly complex embedding* of K. The pair (r_1, r_2) is called the *signature* of the number field K. K has r_1 real embeddings and $2r_2$ properly complex embeddings. If $r_2 = 0$, that is, if all embeddings of K are real, one calls K a *totally real* number field. On the other hand, if $r_1 = 0$, that is, if all embeddings of K are properly complex, then K is called a *totally complex* number field. □

Example 2.30

(1) The number field $\mathbb{Q}[X]/\langle X^2-2\rangle$ is totally real and has the signature $(2, 0)$. (The roots of $X^2 - 2$ are $\pm\sqrt{2}$.)

(2) The number field $\mathbb{Q}[X]/\langle X^2+2\rangle$ is totally complex and has the signature $(0, 1)$. (The roots of $X^2 + 2$ are $\pm\mathrm{i}\sqrt{2}$.)

(3) The number field $K := \mathbb{Q}[X]/\langle X^3-2\rangle$ is neither totally real nor totally complex. The roots of $X^3 - 2$ are $\sqrt[3]{2}$ and $\sqrt[3]{2}\left(\frac{-1\pm\sqrt{-3}}{2}\right)$. The signature of K is $(1, 1)$, that is, K has one real embedding and two properly complex embeddings. □

The simplest examples of number fields are the *quadratic number fields*, that is, number fields of degree 2. Some special properties of quadratic number fields are covered in the exercises. It follows from Exercise 2.136 that every quadratic number field is of the form $\mathbb{Q}(\sqrt{D})$ for some non-zero square-free integer $D \neq 1$.

Now we investigate the $\mathbb{Z}$-module structure of $\mathfrak{O}_K$ for a number field K of degree d. Let $\sigma_1, \ldots, \sigma_d$ be the complex embeddings of K.

Definition 2.103 For an element $\alpha \in K$, we define the *trace* of α (over $\mathbb{Q}$) as

$$\mathrm{Tr}(\alpha) := \sum_{i=1}^{d} \sigma_i(\alpha) \tag{2.15}$$

and the *norm* of α (over $\mathbb{Q}$) as

$$\mathrm{N}(\alpha) := \prod_{i=1}^{d} \sigma_i(\alpha).$$

□

If $g(X)$ is the minimal polynomial of α over $\mathbb{Q}$ and $r := \deg g$, then $r|d$. Moreover, $\prod_{i=1}^{d}(X - \sigma_i(\alpha)) = g(X)^{d/r}$. So $\mathrm{Tr}(\alpha)$ and $\mathrm{N}(\alpha)$ belong to $\mathbb{Q}$. If α is an algebraic integer, then $g(X) \in \mathbb{Z}[X]$, that is, $\mathrm{Tr}(\alpha), \mathrm{N}(\alpha) \in \mathbb{Z}$.

The following properties of the norm and trace functions can be readily verified. Here $\alpha, \beta \in K$ and $c \in \mathbb{Q}$.

$$\begin{aligned}
\mathrm{Tr}(\alpha+\beta) &= \mathrm{Tr}(\alpha) + \mathrm{Tr}(\beta)\,, \\
\mathrm{N}(\alpha\beta) &= \mathrm{N}(\alpha)\,\mathrm{N}(\beta)\,, \\
\mathrm{Tr}(c\alpha) &= c\,\mathrm{Tr}(\alpha)\,, \\
\mathrm{N}(c\alpha) &= c^d\,\mathrm{N}(\alpha)\,, \\
\mathrm{Tr}(c) &= cd\,, \\
\mathrm{N}(c) &= c^d\,.
\end{aligned}$$

Definition 2.104 Let $\beta_1, \ldots, \beta_d \in K$. We call the determinant of the matrix $(\mathrm{Tr}(\beta_i\beta_j))_{1 \leqslant i,j \leqslant d}$, whose ij-th entry is equal to $\mathrm{Tr}(\beta_i\beta_j)$, the *discriminant* $\Delta(\beta_1, \ldots, \beta_d)$ of $\beta_1, \ldots, \beta_d$. Since each $\mathrm{Tr}(\beta_i\beta_j) \in \mathbb{Q}$, it follows that $\Delta(\beta_1, \ldots, \beta_d) \in \mathbb{Q}$. Moreover, if $\beta_1, \ldots, \beta_d$ are all algebraic integers, then $\Delta(\beta_1, \ldots, \beta_d) \in \mathbb{Z}$. □

Proposition 2.46 $\Delta(\beta_1, \ldots, \beta_d) = (\det(\sigma_j(\beta_i)))^2$.

Proof Consider the matrices $D := (\mathrm{Tr}(\beta_i\beta_j))$ and $E := (\sigma_j(\beta_i))$. By definition, we have $\Delta(\beta_1, \ldots, \beta_d) = \det D$. We show that $D = EE^{\mathrm{t}}$, which implies that $\det D = (\det E)^2$. The ij-th entry of EE^{t} is

$$\sum_{k=1}^{d} \sigma_k(\beta_i)\sigma_k(\beta_j) = \sum_{k=1}^{d} \sigma_k(\beta_i\beta_j) = \mathrm{Tr}(\beta_i\beta_j),$$

where the last equality follows from Equation (2.15). ■

Let $K = \mathbb{Q}(\alpha)$ for some $\alpha \in K$ and let $f(X)$ be the minimal polynomial of α over $\mathbb{Q}$. We define the discriminant of f as

$$\Delta(f) := \Delta(1, \alpha, \alpha^2, \ldots, \alpha^{d-1})\,.$$

We have to show that the quantity $\Delta(f)$ is well-defined, that is, independent of the choice of the root α of $f(X)$. Let $\alpha = \alpha_1, \alpha_2, \ldots . \alpha_d$ be all the roots of $f(X)$ and let the complex embedding σ_j of K map α to α_j. By Proposition 2.46, we have $\Delta(f) =$

$(\det E)^2$, where $E = (\sigma_j(\alpha^{i-1})) = (\alpha_j^{i-1})$. Computing the determinant of E gives $\Delta(f) = (-1)^{d(d-1)/2} \prod_{\substack{i,j=1 \\ i\neq j}}^{d} (\alpha_j - \alpha_i)$, which implies that $\Delta(f)$ is independent of the permutations of the conjugates $\alpha_1, \ldots, \alpha_d$ of α. Notice that since $\alpha_1, \ldots, \alpha_d$ are all distinct, $\Delta(f) \neq 0$.

Let us deduce a useful formula for $\Delta(f)$. Write $f(X) = \prod_{i=1}^{d}(X - \alpha_i)$ and take formal derivative to get $f'(X) = \sum_{j=1}^{d} \prod_{\substack{i=1 \\ i\neq j}}^{d}(X - \alpha_i)$, that is, $f'(\alpha_j) = \prod_{\substack{i=1 \\ i\neq j}}^{d}(\alpha_j - \alpha_i)$. Therefore, $\Delta(f) = (-1)^{d(d-1)/2} \prod_{j=1}^{d} f'(\alpha_j) = (-1)^{d(d-1)/2} \prod_{j=1}^{d} \sigma_j(f'(\alpha))$, that is,

$$\Delta(f) = \Delta(1, \alpha, \alpha^2, \ldots, \alpha^{d-1}) = (-1)^{d(d-1)/2}\, \mathrm{N}(f'(\alpha)) \qquad \textbf{(2.16)}$$

For arbitrary $\beta_1, \ldots, \beta_d \in K$, the discriminant $\Delta(\beta_1, \ldots, \beta_d)$ discriminates between the cases that $\beta_1, \ldots, \beta_d$ form a $\mathbb{Q}$-basis of K and that they do not.

Lemma 2.12 Let $\beta_1, \ldots, \beta_d, \gamma_1, \ldots, \gamma_d \in K$ satisfy $\gamma_i = \sum_{k=1}^{d} t_{ik}\beta_k$ for $i = 1, \ldots, d$ and for $t_{ik} \in \mathbb{Q}$. Then $\Delta(\gamma_1, \ldots, \gamma_d) = (\det T)^2 \Delta(\beta_1, \ldots, \beta_d)$, where $T = (t_{ij})$.

Proof Let $E_1 := (\sigma_j(\beta_i))$ and $E_2 := (\sigma_j(\gamma_i))$. Now

$$\sigma_j(\gamma_i) = \sigma_j(\sum_{k=1}^{d} t_{ik}\beta_k) = \sum_{k=1}^{d} t_{ik}\sigma_j(\beta_k)$$

is the ij-th entry of the matrix TE_1, that is, $E_2 = TE_1$. Hence

$$\Delta(\gamma_1, \ldots, \gamma_d) = (\det E_2)^2 = (\det T)^2 (\det E_1)^2 = (\det T)^2 \Delta(\beta_1, \ldots, \beta_d). \quad \blacksquare$$

Corollary 2.19 Let $\mathcal{B}_1 := (\beta_1, \ldots, \beta_d)$ and $\mathcal{B}_2 := (\gamma_1, \ldots, \gamma_d)$ be two $\mathbb{Q}$-bases of K. Let $\Delta(\mathcal{B}_1) := \Delta(\beta_1, \ldots, \beta_d)$ and $\Delta(\mathcal{B}_2) := \Delta(\gamma_1, \ldots, \gamma_d)$. Then $\Delta(\mathcal{B}_2) = (\det T)^2 \Delta(\mathcal{B}_1)$, where T is the change-of-basis matrix from $\mathcal{B}_1$ to $\mathcal{B}_2$. $\blacksquare$

Corollary 2.20 $\beta_1, \ldots, \beta_d \in K$ form a $\mathbb{Q}$-basis of K, if and only if $\Delta(\beta_1, \ldots, \beta_d) \neq 0$.

Proof Let $K = \mathbb{Q}(\alpha)$, $\mathcal{B}_1 := (1, \alpha, \ldots, \alpha^{d-1})$ and $\mathcal{B}_2 := (\beta_1, \ldots, \beta_d)$. Since $\mathcal{B}_1$ is a $\mathbb{Q}$-basis of K, each β_i can be written (uniquely) as $\beta_i = \sum_{j=0}^{d-1} t_{ij}\alpha^j$ with $t_{ij} \in \mathbb{Q}$. By Lemma 2.12, $\Delta(\mathcal{B}_2) = (\det T)^2 \Delta(\mathcal{B}_1)$, where $T := (t_{ij})_{\substack{1 \leqslant i \leqslant d \\ 0 \leqslant j \leqslant d-1}}$. We have seen that $\Delta(\mathcal{B}_1) \neq 0$. Therefore, $\Delta(\mathcal{B}_2) \neq 0 \iff \det T \neq 0 \iff \mathcal{B}_2$ is a $\mathbb{Q}$-basis of K. $\blacksquare$

Finally comes the desired characterization of $\mathfrak{O}_K$.

Theorem 2.55 For a number field K of degree d, the ring $\mathfrak{O}_K$ is a free $\mathbb{Z}$-module of rank d.

Proof Let $\beta_1, \ldots, \beta_d \in K$ form a $\mathbb{Q}$-basis of K. We know that for some $r_1, \ldots, r_d \in \mathbb{Z} \setminus \{0\}$ the elements $r_1\beta_1, \ldots, r_d\beta_d$ are in $\mathfrak{O}_K$ and continue to constitute a $\mathbb{Q}$-basis of

K. So we may assume that the elements $\beta_1, \ldots, \beta_d$ are already in $\mathfrak{O}_K$. Consider the set $\mathcal{S}$ of all $\mathbb{Q}$-bases $(\beta_1, \ldots, \beta_d)$ of K consisting of elements from $\mathfrak{O}_K$ only. By Definition 2.104 and Corollary 2.20, $\Delta(\mathcal{B}) \in \mathbb{Z} \setminus \{0\}$ for every $\mathcal{B} \in \mathcal{S}$. Choose $\mathcal{B} := (\beta_1, \ldots, \beta_d) \in \mathcal{S}$ such that $|\Delta(\mathcal{B})|$ is minimal in $\mathcal{S}$.

Claim: $\mathcal{B}$ is linearly independent over $\mathbb{Z}$.
$\mathcal{B}$ is a $\mathbb{Q}$-basis of K, that is, linearly independent over $\mathbb{Q}$ and so trivially over $\mathbb{Z}$ too.

Claim: $\mathcal{B}$ generates $\mathfrak{O}_K$ as a $\mathbb{Z}$-module.
Assume not, that is, there exists $\alpha \in \mathfrak{O}_K$ such that $\alpha = a_1\beta_1 + \cdots + a_d\beta_d$ with some $a_i \notin \mathbb{Z}$. Without loss of generality, we may assume that $a_1 \notin \mathbb{Z}$ and write $a_1 = a + r$ with $a \in \mathbb{Z}$ and $0 < r < 1$. Define $\gamma_1 := \alpha - a\beta_1 = r\beta_1 + a_2\beta_2 + \cdots + a_d\beta_d$, $\gamma_2 := \beta_2, \ldots, \gamma_d := \beta_d$. Clearly, $\gamma_1, \ldots, \gamma_d \in \mathfrak{O}_K$. Furthermore, if

$$T := \begin{pmatrix} r & a_2 & a_3 & \cdots & a_d \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix},$$

by Lemma 2.12, we have

$$\Delta(\gamma_1, \ldots, \gamma_d) = (\det T)^2 \Delta(\beta_1, \ldots, \beta_d) = r^2 \Delta(\beta_1, \ldots, \beta_d).$$

Since $r \neq 0$, $\Delta(\gamma_1, \ldots, \gamma_d) \neq 0$, that is, $(\gamma_1, \ldots, \gamma_d)$ is again a $\mathbb{Q}$-basis of K (Corollary 2.20), that is, $(\gamma_1, \ldots, \gamma_d) \in \mathcal{S}$. Finally since $r < 1$, we have $|\Delta(\gamma_1, \ldots, \gamma_d)| < |\Delta(\beta_1, \ldots, \beta_d)|$, a contradiction to the choice of $(\beta_1, \ldots, \beta_d)$. Thus every $\alpha \in \mathfrak{O}_K$ has to be a $\mathbb{Z}$-linear combination of $\beta_1, \ldots, \beta_d$. This completes the proof of the second claim and also of the theorem. ■

Definition 2.105 Any $\mathbb{Z}$-basis of $\mathfrak{O}_K$ is called an *integral basis* of K (or of $\mathfrak{O}_K$). □

Corollary 2.21 Every integral basis of K has the same discriminant (for a given K).

Proof Let $\mathcal{B}_1 := (\beta_1, \ldots, \beta_d)$ and $\mathcal{B}_2 := (\gamma_1, \ldots, \gamma_d)$ be two integral bases of K. Let T be the $\mathcal{B}_1$-to-$\mathcal{B}_2$ change-of-basis matrix. $\mathcal{B}_1$ being an integral basis of K, all the entries of T are integers. Also from Corollary 2.19 we have $\Delta(\mathcal{B}_2) = (\det T)^2 \Delta(\mathcal{B}_1)$ and hence $\Delta(\mathcal{B}_1)$ divides and has the same sign as $\Delta(\mathcal{B}_2)$. One can analogously show $\Delta(\mathcal{B}_2) | \Delta(\mathcal{B}_1)$. Therefore, $\Delta(\mathcal{B}_1) = \Delta(\mathcal{B}_2)$. ■

Definition 2.106 Let $\mathcal{B}$ be an integral basis of a number field K. The *discriminant* of K is defined to be the integer $\Delta_K := \Delta(\mathcal{B})$. By Corollary 2.21, Δ_K is well-defined, that is, independent of the choice of the integral basis $\mathcal{B}$ of K. □

Recall that K, as a vector space over $\mathbb{Q}$, always possesses a $\mathbb{Q}$-basis of the form $1, \alpha, \ldots, \alpha^{d-1}$. $\mathfrak{O}_K$, as a $\mathbb{Z}$-module, is free of rank d, but *every number field K need not possess an integral basis of the form* $1, \alpha, \ldots, \alpha^{d-1}$. Whenever it does, $\mathfrak{O}_K = \mathbb{Z}[\alpha]$ is called *monogenic* and an integral basis $1, \alpha, \ldots, \alpha^{d-1}$ of K is called a *power integral basis*. Clearly, if K has a power integral basis $1, \alpha, \ldots, \alpha^{d-1}$, then $K = \mathbb{Q}(\alpha)$. But the converse is not true, that is, for $K = \mathbb{Q}(\alpha)$ with $\alpha \in \mathfrak{O}_K$, $1, \alpha, \ldots, \alpha^{d-1}$ need not be an integral basis of K, even when $\mathfrak{O}_K$ is monogenic.

Example 2.31 Consider the quadratic number field $K := \mathbb{Q}(\sqrt{D})$ for some square-free integer $D \neq 0, 1$. We consider the two cases (See Exercise 2.136):

Case 1: $D \equiv 2, 3 \pmod 4$
Here $\mathfrak{O}_K = \mathbb{Z}[\sqrt{D}]$, that is, $(1, \sqrt{D})$ is a power integral basis of K. The minimal polynomial of $\sqrt{D}$ is $X^2 - D$ and the conjugates of $\sqrt{D}$ are $\pm\sqrt{D}$. Therefore, by Equation (2.16), we have

$$\Delta_K = (-1)^{2(2-1)/2} \,\mathrm{N}(2\sqrt{D}) = -(2\sqrt{D})(-2\sqrt{D}) = 4D.$$

Case 2: $D \equiv 1 \pmod 4$
In this case, $\mathfrak{O}_K = \mathbb{Z}\left[\frac{1+\sqrt{D}}{2}\right]$, that is, $(1, \frac{1+\sqrt{D}}{2})$ is a power integral basis of K. The minimal polynomial of $\frac{1+\sqrt{D}}{2}$ is $X^2 - X - \frac{D-1}{4}$ and the conjugates of $\sqrt{D}$ are $\pm\sqrt{D}$. Therefore, Equation (2.16) gives

$$\begin{aligned} \Delta_K &= (-1)^{2(2-1)/2} \,\mathrm{N}\left(2\left(\frac{1+\sqrt{D}}{2}\right) - 1\right) \\ &= -\,\mathrm{N}(\sqrt{D}) = -(\sqrt{D})(-\sqrt{D}) = D. \end{aligned}$$

□

2.13.3 Unique Factorization of Ideals

Ideals in a number ring possess very rich structures. We prove that number rings are Dedekind domains (Definition 2.99). A Dedekind domain (henceforth abbreviated as DD) need not be a UFD (or a PID). However, it is a ring in which *ideals* admit unique factorizations into products of prime ideals.

Let K be a number field of degree $d = [K : \mathbb{Q}]$ and $\mathfrak{O}_K$ its ring of integers. If $\varphi : A \to B$ is a homomorphism of rings and if $\mathfrak{q}$ is a prime ideal of B, then the contraction $\mathfrak{p} := \varphi^{-1}(\mathfrak{q}) = \mathfrak{q}^c$ is a prime ideal of A. We say that $\mathfrak{q}$ *lies above* or *over* $\mathfrak{p}$. If $A \subseteq B$ and φ is the inclusion homomorphism, then $\mathfrak{p} = A \cap \mathfrak{q}$. For a number field K, we consider the natural inclusion $\mathbb{Z} \hookrightarrow \mathfrak{O}_K$.

Lemma 2.13 Let $\mathfrak{q}$ be a non-zero prime ideal of $\mathfrak{O}_K$. Then $\mathfrak{q}$ lies above a unique non-zero prime ideal of $\mathbb{Z}$. In particular, $\mathfrak{q}$ contains a (unique) rational prime.

Proof Let $\mathfrak{p} := \mathfrak{q} \cap \mathbb{Z}$. If $\mathfrak{p} = 0$, then both $\mathfrak{q}$ and 0 are prime ideals of $\mathfrak{O}_K$ that lie over the zero ideal of $\mathbb{Z}$. Since $0 \subseteq \mathfrak{q}$, $\mathfrak{q} = 0$ by Exercise 2.128(c), a contradiction. ■

Proposition 2.47 $\mathfrak{O}_K$ is Noetherian.

Proof Let $\alpha_1, \ldots, \alpha_d \in \mathfrak{O}_K$ constitute an integral basis of K, that is, $\mathfrak{O}_K = \mathbb{Z}[\alpha_1, \ldots, \alpha_d]$, that is, the ring homomorphism $\mathbb{Z}[X_1, \ldots, X_d] \to \mathfrak{O}_K$ mapping $f(X_1, \ldots, X_d) \mapsto f(\alpha_1, \ldots, \alpha_d)$ is surjective. By Hilbert's basis theorem (Theorem 2.8), the polynomial ring $\mathbb{Z}[X_1, \ldots, X_d]$ is Noetherian and so $\mathfrak{O}_K$, being the quotient of a Noetherian ring (by the isomorphism theorem), is Noetherian too (Example 2.29). ■

Theorem 2.56 The ring $\mathfrak{O}_K$ of integers of a number field K is a Dedekind domain.

Proof We have proved that $\mathfrak{O}_K$ is Noetherian (Proposition 2.47) and integrally closed (Proposition 2.44). It then suffices to show that each non-zero prime ideal $\mathfrak{q}$ of $\mathfrak{O}_K$ is maximal. By Lemma 2.13, $\mathfrak{q}$ lies over a non-zero prime ideal $\mathfrak{p}$ of $\mathbb{Z}$. But $\mathfrak{p}$ is maximal in $\mathbb{Z}$. Exercise 2.128(b) now completes the proof. ■

Now we derive the unique factorization theorem for ideals in a DD. It is going to be a long story. We refer the reader to Definition 2.92 to recall how the product of two ideals is defined.

Lemma 2.14 Let A be a ring, $r \in \mathbb{N}$, $\mathfrak{a}_1, \ldots, \mathfrak{a}_r$ ideals of A, and $\mathfrak{p}$ a prime ideal of A such that $\mathfrak{p} \supseteq \mathfrak{a}_1 \cdots \mathfrak{a}_r$. Then $\mathfrak{p} \supseteq \mathfrak{a}_k$ for some $k \in \{1, \ldots, r\}$. In particular, if A is a DD and $\mathfrak{a}_i$ are non-zero prime ideals, then $\mathfrak{p} = \mathfrak{a}_k$ for some $k \in \{1, \ldots, r\}$.

Proof The proof is obvious for $r = 1$. So assume that $r > 1$. If $\mathfrak{p} \not\supseteq \mathfrak{a}_i$ for all $i = 1, \ldots, r$, then for each i we can choose $a_i \in \mathfrak{a}_i \setminus \mathfrak{p}$ and see that $a_1 \cdots a_r \in \mathfrak{p}$, a contradiction to that $\mathfrak{p}$ is prime. The last statement of the lemma follows from the fact that in a DD every non-zero prime ideal is maximal. ■

We now generalize the concept of ideals.

Definition 2.107 Let A be an integral domain and $K := \mathrm{Q}(A)$. An A-submodule $\mathfrak{a}$ of K is called a *fractional ideal* of A, if $a\mathfrak{a} \subseteq A$ for some $0 \neq a \in A$. □

Every ideal of A is evidently a fractional ideal of A and hence is often called an *integral ideal* of A. Conversely, every fractional ideal of A contained in A is an integral ideal of A. The *principal fractional ideal* Ax is the A-submodule of K generated by $x \in K$. If A is a Noetherian domain, we have the following equivalent characterization of fractional ideals.

Lemma 2.15 Let A be a Noetherian integral domain, $K := \mathrm{Q}(A)$ and $\mathfrak{a} \subseteq K$. Then $\mathfrak{a}$ is a fractional ideal of A, if and only if $\mathfrak{a}$ is a finitely generated A-submodule of K.

Proof [if] Let $\mathfrak{a} = Ax_1 + \cdots + Ax_m$, where $x_i = a_i/b_i$, $a_i, b_i \in A$, $b_i \neq 0$. Then $b_1 \cdots b_m \mathfrak{a} \subseteq A$.
[only if] Let $0 \neq b \in A$ be such that $b\mathfrak{a} \subseteq A$. Now $b\mathfrak{a}$ is an (integral) ideal of A (easy check) and is finitely generated, since A is Noetherian. Let $b\mathfrak{a} = Aa_1 + \cdots + Aa_m$, $a_i \in A$. Then $\mathfrak{a} = Ax_1 + \cdots + Ax_m$, where $x_i := a_i/b \in K$. ■

We define the product of two fractional ideals $\mathfrak{a}, \mathfrak{b}$ of an integral domain A as we did for integral ideals:

$$\mathfrak{ab} := \left\{ \sum_{i=1}^{m} a_i b_i \mid m \in \mathbb{N}_0, a_i \in \mathfrak{a}, b_i \in \mathfrak{b} \right\}.$$

It is easy to check that $\mathfrak{ab}$ is again a fractional ideal of A. Let $\mathcal{F}$ denote the set of non-zero fractional ideals of A. The product of fractional ideals defines a commutative and associative binary operation on $\mathcal{F}$. The ideal A acts as a (multiplicative) identity in $\mathcal{F}$. A fractional ideal $\mathfrak{a}$ of A is called *invertible*, if $\mathfrak{ab} = A$ for some fractional ideal $\mathfrak{b}$ of A. We deduce shortly that if A is a DD, then every non-zero fractional ideal of A is invertible and, therefore, $\mathcal{F}$ is a group under multiplication of fractional ideals.

Lemma 2.16 Let A be a Noetherian domain and $\mathfrak{a}$ an (integral) ideal of A. For some $r \in \mathbb{N}_0$, there exist prime ideals $\mathfrak{q}_1, \ldots, \mathfrak{q}_r$ of A each containing $\mathfrak{a}$ such that $\mathfrak{q}_1 \cdots \mathfrak{q}_r \subseteq \mathfrak{a}$.

Proof Let S be the set of ideals of A for which the lemma does not hold. Assume that $S \neq \emptyset$. Since A is Noetherian, S contains a maximal element, say $\mathfrak{b}$. Clearly, $\mathfrak{b}$ is a proper non-prime ideal of A, that is, for some $a, b \notin \mathfrak{b}$ we have $ab \in \mathfrak{b}$. The ideals $\mathfrak{b}' := \mathfrak{b} + Aa$ and $\mathfrak{b}'' := \mathfrak{b} + Ab$ strictly contain $\mathfrak{b}$ and, therefore, by the maximality of $\mathfrak{b}$ are not in S, that is, there exist prime ideals $\mathfrak{q}'_1, \ldots, \mathfrak{q}'_s$ each containing $\mathfrak{b}'$ (and hence $\mathfrak{b}$) such that $\mathfrak{q}'_1 \cdots \mathfrak{q}'_s \subseteq \mathfrak{b}'$ and prime ideals $\mathfrak{q}''_1, \ldots, \mathfrak{q}''_t$ each containing $\mathfrak{b}''$ (and hence $\mathfrak{b}$) such that $\mathfrak{q}''_1 \cdots \mathfrak{q}''_t \subseteq \mathfrak{b}''$. Moreover, $(\mathfrak{q}'_1 \cdots \mathfrak{q}'_s)(\mathfrak{q}''_1 \cdots \mathfrak{q}''_t) \subseteq \mathfrak{b}'\mathfrak{b}'' = (\mathfrak{b} + Aa)(\mathfrak{b} + Ab) = \mathfrak{b}\mathfrak{b} + a\mathfrak{b} + b\mathfrak{b} + Aab \subseteq \mathfrak{b}$, since $ab \in \mathfrak{b}$, so that $\mathfrak{b} \notin S$, a contradiction. Thus S must be empty. ■

Note that the condition "each containing $\mathfrak{a}$" was necessary in Lemma 2.16 in order to rule out the trivial possibility that $\mathfrak{q}_i = 0$ for some $i \in \{1, \ldots, r\}$.

Lemma 2.17 Let A be a DD, $K := \mathrm{Q}(A)$ and $\mathfrak{q}$ a non-zero prime ideal of A. Define the set

$$\mathfrak{q}^{-1} := \{x \in K \mid x\mathfrak{q} \subseteq A\}\,.$$

Then we have:

(1) $\mathfrak{q}^{-1}$ is a fractional ideal of A.
(2) $A \subsetneqq \mathfrak{q}^{-1}$.
(3) $\mathfrak{q}\mathfrak{q}^{-1} = A$. In particular, every non-zero prime ideal in a DD is invertible.

Proof

(1) Clearly, $\mathfrak{q}^{-1}$ is an A-submodule of K, and for $0 \neq a \in \mathfrak{q}$, we have $a\mathfrak{q}^{-1} \subseteq \mathfrak{q}\mathfrak{q}^{-1} \subseteq A$.

(2) Since $1 \in \mathfrak{q}^{-1}$, we have $A \subseteq \mathfrak{q}^{-1}$. In order to prove the strict inclusion, we take any $0 \neq a \in \mathfrak{q}$ and consider the ideal $\mathfrak{a} := Aa$. By Lemma 2.16, there exist prime ideals $\mathfrak{q}_1, \ldots, \mathfrak{q}_r$ each containing $\mathfrak{a}$ (and hence non-zero) such that $\mathfrak{q}_1 \cdots \mathfrak{q}_r \subseteq \mathfrak{a}$. We choose r to be minimal, so that $\mathfrak{a}$ does not contain the product of any $r-1$ of $\mathfrak{q}_1, \ldots, \mathfrak{q}_r$. Now $\mathfrak{q}_1 \cdots \mathfrak{q}_r \subseteq \mathfrak{q}$ and hence by Lemma 2.14 $\mathfrak{q}_i = \mathfrak{q}$ for some i, say, $i = r$. Choose any $b \in \mathfrak{q}_1 \cdots \mathfrak{q}_{r-1} \setminus \mathfrak{a}$. Since $b \notin \mathfrak{a} = Aa$, we have $b/a \notin A$. On the other hand, $b \in \mathfrak{q}_1 \cdots \mathfrak{q}_{r-1}$ and $\mathfrak{q} = \mathfrak{q}_r$, so that $(b/a)\mathfrak{q} \subseteq (1/a)(\mathfrak{q}_1 \cdots \mathfrak{q}_r) \subseteq (1/a)\mathfrak{a} = (1/a)(Aa) = A$, that is, $b/a \in \mathfrak{q}^{-1} \setminus A$.

(3) By the definition of $\mathfrak{q}^{-1}$, it follows that $\mathfrak{q}\mathfrak{q}^{-1}$ is contained in and hence an integral ideal of A. Since $A \subseteq \mathfrak{q}^{-1}$, it follows that $\mathfrak{q} = \mathfrak{q}A \subseteq \mathfrak{q}\mathfrak{q}^{-1}$. Since $\mathfrak{q}$ is a maximal ideal, we then have $\mathfrak{q}\mathfrak{q}^{-1} = \mathfrak{q}$ or $\mathfrak{q}\mathfrak{q}^{-1} = A$. Assume that $\mathfrak{q}\mathfrak{q}^{-1} = \mathfrak{q}$. We claim that this assumption implies that $\mathfrak{q}^{-1} \subseteq A$, a contradiction to Part (2). So we must have $\mathfrak{q}\mathfrak{q}^{-1} = A$. For proving the claim, let $b \in \mathfrak{q}^{-1}$ and choose $0 \neq a \in \mathfrak{q}$. Then we have $ab \in \mathfrak{q}\mathfrak{q}^{-1} = \mathfrak{q}$ and, therefore, $ab^2 = (ab)b \in \mathfrak{q}$, $ab^3 = (ab^2)b \in \mathfrak{q}$ and so on. For each $m \in \mathbb{N}_0$, define the ideal $\mathfrak{a}_m := \sum_{i=0}^{m} Aab^i$. Then $\mathfrak{a}_0 \subseteq \mathfrak{a}_1 \subseteq \mathfrak{a}_2 \subseteq \cdots$ is an ascending chain of ideals in A. Since A is Noetherian, the chain must be stationary, that is, for some $m \in \mathbb{N}$ we have $\mathfrak{a}_m = \mathfrak{a}_{m-1}$, that is, $ab^m \in \sum_{i=0}^{m-1} Aab^i$, that is, $ab^m = \sum_{i=0}^{m-1} a_i ab^i$ with $a_i \in A$. Since A is an integral domain and $a \neq 0$, we see that b is integral over A. Since A is integrally closed, $b \in A$. Therefore, $\mathfrak{q}^{-1} \subseteq A$, as claimed. ■

Theorem 2.57 Every non-zero ideal $\mathfrak{a}$ in a DD A can be represented as a product of prime ideals of A. Moreover, such a factorization of $\mathfrak{a}$ is unique up to permutations of the factors.

Proof If $\mathfrak{a} = A$, there is nothing to prove. So let $\mathfrak{a}$ be a proper ideal of A. We first show that if $\mathfrak{a}$ contains a product of non-zero prime ideals, then $\mathfrak{a}$ *is* a product of prime ideals. By Lemma 2.16, we have prime ideals $\mathfrak{q}_1, \ldots, \mathfrak{q}_r$, $r \in \mathbb{N}$, of A each containing $\mathfrak{a}$, such that $\mathfrak{q}_1 \cdots \mathfrak{q}_r \subseteq \mathfrak{a}$. Let us choose r to be minimal and proceed by induction on r. If $r = 1$, $\mathfrak{a} = \mathfrak{q}_1$ is already prime. So take $r > 1$ and assume that if an ideal $\mathfrak{b}$ of A contains a product of $r-1$ or less non-zero prime ideals of A, then $\mathfrak{b}$ is a product of prime ideals. Let $\mathfrak{q}$ be a maximal ideal containing $\mathfrak{a}$. We then have $\mathfrak{q}_1 \cdots \mathfrak{q}_r \subseteq \mathfrak{a} \subseteq \mathfrak{q}$ and by Lemma 2.14 $\mathfrak{q} = \mathfrak{q}_i$ for some i, say, $i = r$. Now, consider the fractional ideal $\mathfrak{b} := \mathfrak{a}\mathfrak{q}_r^{-1}$. Then $\mathfrak{b} \subseteq \mathfrak{q}_r\mathfrak{q}_r^{-1} = A$ and so $\mathfrak{b}$ is an integral ideal of A. Furthermore $\mathfrak{b} = \mathfrak{a}\mathfrak{q}_r^{-1} \supseteq \mathfrak{q}_1 \cdots \mathfrak{q}_r\mathfrak{q}_r^{-1} = (\mathfrak{q}_1 \cdots \mathfrak{q}_{r-1})A = \mathfrak{q}_1 \cdots \mathfrak{q}_{r-1}$, that is, $\mathfrak{b}$ contains a product of $r-1$ non-zero prime ideals. By the induction hypothesis, $\mathfrak{b}$ is a product of prime ideals, that is, $\mathfrak{b} = \mathfrak{q}_1' \cdots \mathfrak{q}_s'$. But then $\mathfrak{a} = \mathfrak{a}A = \mathfrak{a}(\mathfrak{q}_r^{-1}\mathfrak{q}_r) = (\mathfrak{a}\mathfrak{q}_r^{-1})\mathfrak{q}_r = \mathfrak{b}\mathfrak{q}_r = \mathfrak{q}_1' \cdots \mathfrak{q}_s'\mathfrak{q}_r$ is also a product of prime ideals.

In order to prove the uniqueness of this product, let $\mathfrak{a} = \mathfrak{q}_1 \cdots \mathfrak{q}_r = \mathfrak{p}_1 \cdots \mathfrak{p}_t$ with prime ideals $\mathfrak{q}_i$ and $\mathfrak{p}_j$. Now $\mathfrak{q}_1 \supseteq \mathfrak{q}_1 \cdots \mathfrak{q}_r = \mathfrak{p}_1 \cdots \mathfrak{p}_t$ and by Lemma 2.14 $\mathfrak{q}_1 = \mathfrak{p}_j$ for some $j \in \{1, \ldots, t\}$, say, $j = 1$. Then $\mathfrak{q}_2 \cdots \mathfrak{q}_r = A\mathfrak{q}_2 \cdots \mathfrak{q}_r = (\mathfrak{q}_1^{-1}\mathfrak{q}_1)\mathfrak{q}_2 \cdots \mathfrak{q}_r = \mathfrak{q}_1^{-1}(\mathfrak{q}_1 \cdots \mathfrak{q}_r) = \mathfrak{p}_1^{-1}(\mathfrak{p}_1 \cdots \mathfrak{p}_t) = \mathfrak{p}_2 \cdots \mathfrak{p}_t$. Proceeding in this way shows the desired uniqueness. ■

In the factorization of a non-zero ideal of a DD, we do not rule out the possibility of repeated occurrences of factors. Taking this into account shows that every non-zero ideal $\mathfrak{a}$ in a DD A admits a *unique* factorization

$$\mathfrak{a} = \mathfrak{q}_1^{e_1} \cdots \mathfrak{q}_r^{e_r}$$

with *distinct* non-zero prime ideals $\mathfrak{q}_1, \ldots, \mathfrak{q}_r$ and with exponents $e_1, \ldots, e_r \in \mathbb{N}$. Here uniqueness is up to permutations of the indexes $1, \ldots, r$. This factorization can be extended to fractional ideals, but this time we have to allow non-positive exponents. First note that for integers $e_1, \ldots, e_r$ and non-zero prime ideals $\mathfrak{q}_1, \ldots, \mathfrak{q}_r$ of A the product $\mathfrak{a} := \mathfrak{q}_1^{e_1} \cdots \mathfrak{q}_r^{e_r}$ is well-defined and is a fractional ideal of $\mathfrak{a}$. The converse is proved in the following corollary.

Corollary 2.22 Every non-zero fractional ideal $\mathfrak{a}$ of a DD A admits a unique factorization of the form $\mathfrak{a} = \mathfrak{q}_1^{e_1} \cdots \mathfrak{q}_r^{e_r}$ with non-zero prime ideals $\mathfrak{q}_1, \ldots, \mathfrak{q}_r$ of A and with exponents $e_i \in \mathbb{Z}$. Moreover for such a fractional ideal $\mathfrak{a}$ we have $\mathfrak{a}(\mathfrak{q}_1^{-e_1} \cdots \mathfrak{q}_r^{-e_r}) = A$.

Proof By definition, there exists $0 \neq a \in A$ such that $a\mathfrak{a} \subseteq A$. But then $a\mathfrak{a} = (Aa)\mathfrak{a}$ is an integral ideal of A. We write $a\mathfrak{a} = \prod_{i=1}^{r} \mathfrak{q}_i^{f_i}$ and $Aa = \prod_{i=1}^{r} \mathfrak{q}_i^{g_i}$ with $f_i, g_i \in \mathbb{N}_0$. Since each non-zero prime ideal is invertible (Lemma 2.17(3)), it follows that $\mathfrak{a} = \prod_{i=1}^{r} \mathfrak{q}_i^{f_i - g_i}$. This proves the existence of a factorization of $\mathfrak{a}$. The proof for the uniqueness is left to the reader as an easy exercise. The last assertion follows from a repeated use of Lemma 2.17(3). ■

The fractional ideal $\mathfrak{q}_1^{-e_1} \cdots \mathfrak{q}_r^{-e_r}$ in Corollary 2.22 is denoted by $\mathfrak{a}^{-1}$. We have $\mathfrak{a}^{-1}\mathfrak{a} = \mathfrak{a}\mathfrak{a}^{-1} = A$. One can easily verify that $\mathfrak{a}^{-1}$ defined as above is equal to the set

$$\mathfrak{a}^{-1} = \{x \in K \mid x\mathfrak{a} \subseteq A\}.$$

In fact, one can use the last equality as the definition for $\mathfrak{a}^{-1}$.

To sum up, every non-zero fractional ideal of a DD A is invertible and the set $\mathcal{F}$ of all non-zero fractional ideals of A is a group. The unit ideal A acts as the identity in $\mathcal{F}$.

As in every group, we have the cancellation law(s) in $\mathcal{F}$.

Corollary 2.23 Let A be a DD and $\mathfrak{a}, \mathfrak{b}, \mathfrak{c}$ fractional ideals of A. If $\mathfrak{ac} = \mathfrak{bc}$ and $\mathfrak{c} \neq 0$, then $\mathfrak{a} = \mathfrak{b}$. ■

In view of unique factorization of ideals in A, we can speak of the divisibility of integral ideals in A. Let $\mathfrak{a}$ and $\mathfrak{b}$ be two integral ideals of A. We say that $\mathfrak{a}$ *divides* $\mathfrak{b}$ and write $\mathfrak{a}|\mathfrak{b}$, if $\mathfrak{b} = \mathfrak{ac}$ for some *integral* ideal $\mathfrak{c}$ of A. We now show that the condition $\mathfrak{a}|\mathfrak{b}$ is equivalent to the condition $\mathfrak{a} \supseteq \mathfrak{b}$. Thus for ideals in a DD the term *divides* is synonymous with *contains*.

Corollary 2.24 Let $\mathfrak{a}$ and $\mathfrak{b}$ be integral ideals of a DD A. Then $\mathfrak{a}|\mathfrak{b}$ if and only if $\mathfrak{a} \supseteq \mathfrak{b}$.

Proof [if] If $\mathfrak{a} \supseteq \mathfrak{b}$, we have $A = \mathfrak{a}^{-1}\mathfrak{a} \supseteq \mathfrak{a}^{-1}\mathfrak{b}$, that is, $\mathfrak{c} := \mathfrak{a}^{-1}\mathfrak{b}$ is an integral ideal of A. Also $\mathfrak{b} = A\mathfrak{b} = (\mathfrak{a}\mathfrak{a}^{-1})\mathfrak{b} = \mathfrak{a}(\mathfrak{a}^{-1}\mathfrak{b}) = \mathfrak{ac}$.
[only if] If $\mathfrak{b} = \mathfrak{ac}$ for some integral ideal $\mathfrak{c}$, we have $\mathfrak{b} \subseteq \mathfrak{a} \cap \mathfrak{c} \subseteq \mathfrak{a}$. ■

Corollary 2.25 Let $\mathfrak{a} := \mathfrak{q}_1^{e_1} \cdots \mathfrak{q}_r^{e_r}$ and $\mathfrak{b} := \mathfrak{q}_1^{f_1} \cdots \mathfrak{q}_r^{f_r}$ with $e_i, f_i \in \mathbb{N}_0$ be the prime decompositions of two non-zero integral ideals of a DD A. Then $\mathfrak{a}|\mathfrak{b}$ if and only if $e_i \leqslant f_i$ for all $i = 1, \ldots, r$.

Proof [if] We have $\mathfrak{b} = \mathfrak{ac}$, where $\mathfrak{c} := \mathfrak{q}_1^{f_1 - e_1} \cdots \mathfrak{q}_r^{f_r - e_r}$ is an integral ideal of A.
[only if] Let $\mathfrak{b} = \mathfrak{ac}$ for some integral ideal $\mathfrak{c}$ of A. Clearly, $\mathfrak{c} \neq 0$ and we can write the prime decomposition $\mathfrak{c} = \mathfrak{q}_1^{l_1} \cdots \mathfrak{q}_r^{l_r} \mathfrak{q}_{r+1}^{l_{r+1}} \cdots \mathfrak{q}_{r+s}^{l_{r+s}}$ with $l_i \geqslant 0$. We have $\mathfrak{q}_1^{f_1} \cdots \mathfrak{q}_r^{f_r} = \mathfrak{q}_1^{e_1+l_1} \cdots \mathfrak{q}_r^{e_r+l_r} \mathfrak{q}_{r+1}^{l_{r+1}} \cdots \mathfrak{q}_{r+s}^{l_{r+s}}$. By unique factorization, we have $f_1 = e_1 + l_1, \ldots, f_r = e_r + l_r$ and $l_{r+1} = \cdots = l_{r+s} = 0$. ■

As we pass from $\mathbb{Z}$ to $\mathfrak{O}_K$, the notion of unique factorization passes from the element level to the ideal level. If a DD is already a PID, these two concepts are equivalent. (Non-zero prime ideals in a PID are generated by prime elements.) Though every UFD need not be a PID, we have the following result for a DD.

Proposition 2.48 A Dedekind domain A is a UFD, if and only if A is a PID.

Proof [if] Every PID is a UFD (Theorem 2.11).
[only if] Let A be a UFD. In order to show that A is a PID, it suffices (in view of Theorem 2.57) to show that every non-zero prime ideal $\mathfrak{q}$ of A is a principal ideal. Choose any non-zero $a \in \mathfrak{q}$. Then $\mathfrak{q} \supseteq \langle a \rangle$. Now a is a non-unit in A (since otherwise we would have $\mathfrak{q} = A$) and A is assumed to be a UFD. Thus we can write $a = uq_1 \cdots q_r$ for $r \in \mathbb{N}$, $u \in A^*$ and for prime elements q_i in A. Clearly, each $\langle q_i \rangle$ is a non-zero prime ideal of A and $\langle a \rangle = \langle q_1 \rangle \cdots \langle q_r \rangle$. Therefore, $\mathfrak{q} \supseteq \langle q_1 \rangle \cdots \langle q_r \rangle$ and hence by Lemma 2.14 $\mathfrak{q} = \langle q_i \rangle$ for some $i \in \{1, \ldots, r\}$. ■

In the rest of this section, we abbreviate $\mathfrak{O}_K$ as $\mathfrak{O}$, if K is implicit in the context.

2.13.4 Norms of Ideals

We have seen that the ring $\mathfrak{O} = \mathfrak{O}_K$ is a free $\mathbb{Z}$-module of rank d. The same result holds for every non-zero ideal $\mathfrak{a}$ of $\mathfrak{O}$. Let $\beta_1, \ldots, \beta_d$ constitute an integral basis of K.

One can choose rational integers a_{ij} with each a_{ii} positive such that

$$\begin{aligned} \gamma_1 &:= a_{11}\beta_1 \\ \gamma_2 &:= a_{21}\beta_1 + a_{22}\beta_2 \\ &\vdots \\ \gamma_d &:= a_{d1}\beta_1 + a_{d2}\beta_2 + \cdots + a_{dd}\beta_d \end{aligned} \tag{2.17}$$

constitute a $\mathbb{Z}$-basis of $\mathfrak{a}$. Moreover, the discriminant $\Delta(\gamma_1, \ldots, \gamma_d)$ is independent of the choice of an integral basis $\gamma_1, \ldots, \gamma_d$ of $\mathfrak{a}$ and is called the *discriminant* of $\mathfrak{a}$, denoted $\Delta(\mathfrak{a})$. It follows that $\mathfrak{a}$ can be generated as an ideal (that is, as an $\mathfrak{O}$-module) by at most d elements. We omit the proof of the following tighter result.

Proposition 2.49 Every (integral) ideal in a DD A is generated by (at most) two elements. More precisely, for a proper non-zero ideal $\mathfrak{a}$ of A and for any $0 \neq a \in \mathfrak{a}$ there exists $b \in \mathfrak{a}$ with $\mathfrak{a} = \langle a, b\rangle = \langle a\rangle + \langle b\rangle$. ■

Definition 2.108 The *norm* $\mathrm{N}(\mathfrak{a})$ of a non-zero ideal $\mathfrak{a}$ of $\mathfrak{O}$ is defined as the cardinality of the quotient ring $\mathfrak{O}/\mathfrak{a}$. It is customary to define the norm of the zero ideal as zero. □

Using the integers a_{ij} of Equations (2.17), we can write

$$\mathrm{N}(\mathfrak{a}) = a_{11} \cdots a_{dd} = \sqrt{\Delta(\mathfrak{a})/\Delta_K}\,. \tag{2.18}$$

Corollary 2.26 For every non-zero ideal $\mathfrak{a}$ of $\mathfrak{O}$, the quotient ring $\mathfrak{O}/\mathfrak{a}$ is a finite ring. In particular, if $\mathfrak{q}$ is a non-zero prime (hence maximal) ideal of $\mathfrak{O}$, then $\mathfrak{O}/\mathfrak{q}$ is a finite field. ■

It is tempting to define the norm of an element $\alpha \in \mathfrak{O}$ to be the norm of the principal ideal $\langle\alpha\rangle = \mathfrak{O}\alpha$. It turns out that this new definition is (almost) the same as the old definition of $\mathrm{N}(\alpha)$. More precisely:

Proposition 2.50 For any element $\alpha \in \mathfrak{O}$, we have $\mathrm{N}(\langle\alpha\rangle) = |\mathrm{N}(\alpha)|$.

Proof The result is obvious for $\alpha = 0$. So assume that $\alpha \neq 0$ and call $\mathfrak{a} := \langle\alpha\rangle$. Let $\beta_1, \ldots, \beta_d$ be an integral basis of $\mathfrak{O}$. It is an easy check that $\alpha\beta_1, \ldots, \alpha\beta_d$ is an integral basis of $\mathfrak{a}$. Let $\sigma_1, \ldots, \sigma_d$ be the complex embeddings of K. Then $\Delta(\mathfrak{a})$ is the square of the determinant of the matrix

$$(\sigma_j(\alpha\beta_i)) = (\sigma_j(\alpha)\sigma_j(\beta_i)) = \begin{pmatrix} \sigma_1(\alpha) & 0 & \cdots & 0 \\ 0 & \sigma_2(\alpha) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_d(\alpha) \end{pmatrix} (\sigma_j(\beta_i))\,.$$

It follows that $\Delta(\mathfrak{a}) = \mathrm{N}(\alpha)^2\Delta(\beta_1, \ldots, \beta_d) = \mathrm{N}(\alpha)^2\Delta_K$. Equation (2.18) now completes the proof. ■

Corollary 2.27 For any $a \in \mathbb{Z}$, we have $\mathrm{N}(\mathfrak{O}a) = |a^d|$. ■

Like the norm of elements, the norm of ideals is also multiplicative. We omit the (not-so-difficult) proof here.

Proposition 2.51 Let $\mathfrak{a}$ and $\mathfrak{b}$ be ideals in $\mathfrak{O}$. Then, $\mathrm{N}(\mathfrak{ab}) = \mathrm{N}(\mathfrak{a})\,\mathrm{N}(\mathfrak{b})$. ■

The following immediate corollary often comes handy.

Corollary 2.28 Let $\mathfrak{a}$ and $\mathfrak{b}$ be non-zero ideals of $\mathfrak{O}$. If $\mathfrak{a} = \mathfrak{q}_1^{e_1} \cdots \mathfrak{q}_r^{e_r}$ is the factorization of $\mathfrak{a}$, then $\mathrm{N}(\mathfrak{a}) = \mathrm{N}(\mathfrak{q}_1)^{e_1} \cdots \mathrm{N}(\mathfrak{q}_r)^{e_r}$. In particular, if $\mathfrak{a}|\mathfrak{b}$, then $\mathrm{N}(\mathfrak{a})|\,\mathrm{N}(\mathfrak{b})$ (in $\mathbb{Z}$). ■

2.13.5 Rational Primes in Number Rings

The behaviour of rational primes in number rings is an interesting topic of study in algebraic number theory. Let K be a number field of degree d and $\mathfrak{O} := \mathfrak{O}_K$. Consider a rational prime p and denote by $\langle p\rangle$ the ideal $\mathfrak{O}p$ generated by p in $\mathfrak{O}$. We use the symbol $\mathfrak{p}$ to denote the (prime) ideal of $\mathbb{Z}$ generated by p. Further let

$$\langle p\rangle = \mathfrak{q}_1^{e_1} \cdots \mathfrak{q}_r^{e_r} \tag{2.19}$$

be the prime factorization of $\langle p\rangle$ with $r \in \mathbb{N}$, with pairwise distinct non-zero prime ideals $\mathfrak{q}_i$ of $\mathfrak{O}$ and with $e_i \in \mathbb{N}$. For each i, we have $\langle p\rangle \subseteq \mathfrak{q}_i$, that is, $p \in \mathfrak{q}_i$, that is, $\mathfrak{p} = \mathbb{Z} \cap \mathfrak{q}_i$ (Lemma 2.13), that is, $\mathfrak{q}_i$ lies over $\mathfrak{p}$. Conversely if $\mathfrak{q}$ is an ideal of $\mathfrak{O}$ lying over $\mathfrak{p}$, then $p \in \mathfrak{q}$, that is, $\langle p\rangle \subseteq \mathfrak{q}$, that is, $\mathfrak{q}|\mathfrak{q}_1^{e_1} \cdots \mathfrak{q}_r^{e_r}$, that is, $\mathfrak{q} = \mathfrak{q}_i$ for some i. Thus, $\mathfrak{q}_1, \ldots, \mathfrak{q}_r$ are precisely all the prime ideals of $\mathfrak{O}$ that lie over $\mathfrak{p}$.

By Corollary 2.27, $\mathrm{N}(\langle p\rangle) = p^d$. By Corollary 2.28, each $\mathrm{N}(\mathfrak{q}_i)$ divides p^d and is again a power p^{d_i} of p.

Definition 2.109 We define the *ramification index* of $\mathfrak{q}_i$ over p (or $\mathfrak{p}$) as $e(\mathfrak{q}_i/p) := e_i$. This is the largest $e \in \mathbb{N}$ such that $\mathfrak{q}_i^e$ divides (that is, contains) $\langle p\rangle$. The integer d_i (where $\mathrm{N}(\mathfrak{q}_i) = p^{d_i}$) is called the *inertial degree* $d(\mathfrak{q}_i/p)$ of $\mathfrak{q}_i$ over p. □

By the multiplicative property of norms, we have

$$d = \sum_{i=1}^{r} e_i d_i = \sum_{i=1}^{r} e(\mathfrak{q}_i/p) d(\mathfrak{q}_i/p)\,.$$

Definition 2.110 If $r = d$, so that each $e_i = d_i = 1$, we say that the prime p (or $\mathfrak{p}$) *splits completely* in $\mathfrak{O}$. On the other extreme, if $r = 1$, $e_1 = 1$, $d_1 = d$, then $\langle p\rangle$ is prime in $\mathfrak{O}$ and we say that p is *inert* in $\mathfrak{O}$. Finally, if $e_i > 1$ for some i, we say that the prime p *ramifies* in $\mathfrak{O}$. If $r = 1$ and $e_1 = d$ (so that $d_1 = 1$), then the prime p is said to be *totally ramified* in $\mathfrak{O}$. □

The following important result is due to Dedekind. Its proof is long and complicated and is omitted here.

Theorem 2.58 A rational prime p ramifies in $\mathfrak{O}_K$, if and only if p divides the discriminant Δ_K. In particular, there are only finitely many rational primes that ramify in $\mathfrak{O}_K$. ■

Though this is not the case in general, let us assume that the ring $\mathfrak{O}$ is monogenic (that is, $\mathfrak{O} = \mathbb{Z}[\alpha]$ for some $\alpha \in \mathfrak{O}$) and try to compute the explicit factorization (Equality (2.19)) of $\langle p\rangle$ in $\mathfrak{O}$. In this case, $K = \mathbb{Q}(\alpha)$ and let $f(X) \in \mathbb{Z}[X]$ be the minimal polynomial of α. We then have $\mathfrak{O} = \mathbb{Z}[\alpha] \cong \mathbb{Z}[X]/\langle f(X)\rangle$.

Let us agree to write the canonical image of any polynomial $g(X) \in \mathbb{Z}[X]$ in $\mathbb{F}_p[X] = \mathbb{Z}[X]/p\mathbb{Z}[X]$ as $\bar{g}(X)$. We write the factorization of $\bar{f}(X)$ as

$$\bar{f}(X) = \bar{f}_1(X)^{e_1} \cdots \bar{f}_r(X)^{e_r}$$

with $r, e_1, \ldots, e_r \in \mathbb{N}$ and with pairwise distinct irreducible polynomials $\bar{f}_i \in \mathbb{F}_p[X]$. If $d_i := \deg \bar{f}_i$, then $\sum_{i=1}^r e_i d_i = d$. For each $i = 1, \ldots, r$ choose $f_i(X) \in \mathbb{Z}[X]$ whose reduction modulo p is $\bar{f}_i(X)$. Define the ideals

$$\mathfrak{q}_i := \langle p, f_i(\alpha) \rangle$$

of $\mathfrak{O}$. Since $\mathfrak{O} \cong \mathbb{Z}[X]/\langle f(X) \rangle$, we have

$$\mathfrak{O}/\langle p \rangle \cong \mathbb{Z}[X]/\langle p, f(X) \rangle \cong \mathbb{F}_p[X]/\langle \bar{f}(X) \rangle$$

and

$$\mathfrak{O}/\mathfrak{q}_i = \mathfrak{O}/\langle p, f_i(\alpha) \rangle \cong \mathbb{Z}[X]/\langle p, f(X), f_i(X) \rangle \cong \mathbb{F}_p[X]/\langle \bar{f}_i(X) \rangle \cong \mathbb{F}_{p^{d_i}}.$$

Therefore, $\mathfrak{q}_i$ are non-zero prime ideals of $\mathfrak{O}$ with $\mathrm{N}(\mathfrak{q}_i) = p^{d_i}$. Thus $\mathrm{N}(\langle p \rangle) = p^d = p^{\sum_{i=1}^r e_i d_i} = \mathrm{N}(\mathfrak{q}_1)^{e_1} \cdots \mathrm{N}(\mathfrak{q}_r)^{e_r} = \mathrm{N}(\mathfrak{q}_1^{e_1} \cdots \mathfrak{q}_r^{e_r})$. On the other hand, $\mathfrak{q}_1^{e_1} \cdots \mathfrak{q}_r^{e_r} = \prod_{i=1}^r \langle p, f_i(\alpha) \rangle^{e_i} \subseteq \langle p \rangle$, since $f(\alpha) = 0$ and $f(X) - \prod_{i=1}^r f_i(X)^{e_i} \in p\mathbb{Z}[X]$. Thus we must have $\langle p \rangle = \mathfrak{q}_1^{e_1} \cdots \mathfrak{q}_r^{e_r}$, that is, we have obtained the desired factorization of $\langle p \rangle$.

Let us now concentrate on an example of this explicit factorization.

Example 2.32 Let $D \neq 0, 1$ be a square-free integer congruent to 2 or 3 modulo 4. If $K := \mathbb{Q}(\sqrt{D})$, then $\mathfrak{O} = \mathbb{Z}[\sqrt{D}\,]$ is monogenic. We take an odd rational prime p and compute the factorization of $\langle p \rangle$ in $\mathfrak{O}$. We have to factorize modulo p the minimal polynomial $f(X) := X^2 - D$. We consider three cases separately based on the value of the Legendre symbol $\left(\frac{D}{p}\right)$.

Case 1: $\left(\frac{D}{p}\right) = 0$

In this case, $p|D$, that is, $\bar{f}(X) = X^2$. Then $\langle p \rangle = \mathfrak{q}^2$, where $\mathfrak{q} := \langle p, \sqrt{D} \rangle$. Thus p (totally) ramifies in $\mathfrak{O}$.

Case 2: $\left(\frac{D}{p}\right) = 1$

Since p is assumed to be an odd prime, the two square roots of D modulo p are distinct. Let δ be an integer with $\delta^2 \equiv D \pmod{p}$. Then $\bar{f}(X) = (X - \delta)(X + \delta)$. In this case, $\langle p \rangle = \mathfrak{q}_1 \mathfrak{q}_2$, where $\mathfrak{q}_1 := \langle p, \sqrt{D} - \delta \rangle$ and $\mathfrak{q}_2 := \langle p, \sqrt{D} + \delta \rangle$. Thus p splits (completely) in $\mathfrak{O}$.

Case 3: $\left(\frac{D}{p}\right) = -1$

The polynomial $\bar{f}(X) = X^2 - D$ is irreducible in $\mathbb{F}_p[X]$ and hence $\langle p, f(\sqrt{D}) \rangle = \langle p, 0 \rangle = \langle p \rangle$ remains prime in $\mathfrak{O}$, that is, p is inert in $\mathfrak{O}$.

Thus the quadratic residuosity of D modulo p dictates the behaviour of p in $\mathfrak{O}$.

Let us finally look at the fate of the even prime 2 in $\mathfrak{O}$. If D is even, then $\bar{f}(X) = X^2$ and if D is odd, then $\bar{f}(X) = (X + 1)^2$. In each case, 2 ramifies in $\mathfrak{O}$.

Recall from Example 2.31 that $\Delta_K = 4D$. Thus we have a confirmation of the fact that a rational prime p ramifies in $\mathfrak{O}$ if and only if $p|\Delta_K$. □

One can similarly study the behaviour of rational primes in

$$\mathfrak{O}_{\mathbb{Q}(\sqrt{D})} = \mathbb{Z}[(1+\sqrt{D})/2],$$

where $D \equiv 1 \pmod 4$ is a square-free integer $\neq 0, 1$.

2.13.6 Units in a Number Ring

There are just two units in $\mathbb{Z}$, namely ± 1. In a general number ring, there may be many more units. For example, all the units in the ring $\mathbb{Z}[\mathrm{i}]$ of Gaussian integers are $\pm 1, \pm \mathrm{i}$. There may even be an infinite number of units in a number ring. It can be shown that $\pm(1+\sqrt{2})^n$, $n \in \mathbb{Z}$, are all the units of $\mathbb{Z}[\sqrt{2}]$. (Note that for all $n \neq 0$ the absolute values of $\pm(1+\sqrt{2})^n$ are different from 1.) $\mathbb{Z}[\sqrt{2}]$ is a PID. So we can think of factorizations in $\mathfrak{O} := \mathfrak{O}_{\mathbb{Z}[\sqrt{2}]}$ as element-wise factorizations. To start with, we fix a set of pairwise non-associate prime *elements* of $\mathfrak{O}$. Every non-zero element of $\mathfrak{O}$ admits a factorization $up_1^{e_1} \cdots p_r^{e_r}$ for prime "representatives" p_i and for a unit u of the form $\pm(1+\sqrt{2})^n$. Thus, in order to complete the picture of factorization, we need machinery to handle the units in a number ring.

Let K be a number field of degree d and signature (r_1, r_2). We have $d = r_1 + 2r_2$. The set of units in $\mathfrak{O} := \mathfrak{O}_K$ is denoted by $\mathfrak{U} := \mathfrak{U}_K := \mathfrak{O}^*$. We know that $\mathfrak{U}$ is an (Abelian) group under (complex) multiplication. Our basic aim now is to reveal the structure of the group $\mathfrak{U}$.

Every Abelian group is a $\mathbb{Z}$-module and, if finitely generated and not free, contains torsion elements, that is, (non-identity) elements of finite order > 1.[19] $\mathfrak{U}$ always contains the element -1 of order 2. The torsion subgroup of $\mathfrak{U}$ is denoted by $\mathfrak{R} := \mathfrak{R}_K := \mathfrak{U}_{\mathrm{tors}}$. We have $\mathfrak{U} \cong \mathfrak{R} \times \mathfrak{G}$, where $\mathfrak{G}$ is a torsion-free group. It turns out that $\mathfrak{R}$ is a finite group (and hence cyclic) and that $\mathfrak{G}$ is finitely generated and hence free, that is, $\mathfrak{G} \cong \mathbb{Z}^\rho$ for some $\rho \in \mathbb{N}_0$. From *Dirichlet's unit theorem* (which we do not prove), it follows that $\rho = r_1 + r_2 - 1$. Thus, $\mathfrak{G}$ has a $\mathbb{Z}$-basis consisting of ρ elements, say $\xi_1, \ldots, \xi_\rho$, and every unit of $\mathfrak{U}$ can be uniquely expressed as $\omega \xi_1^{e_1} \cdots \xi_\rho^{e_\rho}$, where ω is a root of unity and $e_i \in \mathbb{Z}$. A set of generators of $\mathfrak{G}$ is called a *set of fundamental units*.

Example 2.33 Let $D \neq 0, 1$ be a square-free integer, $K := \mathbb{Q}(\sqrt{D})$ and $\mathfrak{O} := \mathfrak{O}_K$. If $D < 0$, the signature of K is $(0, 1)$ and the value of ρ for $\mathfrak{G}$ is $0 + 1 - 1 = 0$, that is, $\mathfrak{U} = \mathfrak{R}$, that is, $\mathfrak{U}$ is finite in this case.

Now, suppose $D > 0$. K is a real field in this case, so that $\mathfrak{R} = \{1, -1\}$. Also the signature of K is $(2, 0)$, that is, $\rho = 2 + 0 - 1 = 1$. This means that $\mathfrak{O}$ contains an infinite number of units. Let ξ be a fundamental unit of $\mathfrak{O}$. Then, every unit of $\mathfrak{O}$ is of the form $\pm\xi^n$, $n \in \mathbb{Z}$. □

EXERCISE SET 2.13

2.126 (a) If $A \subseteq B$ and $B \subseteq C$ are integral extensions of rings, show that $A \subseteq C$ is also an integral extension.

[19] Every finitely generated torsion-free module over a PID is free.

(b) Let $A \subseteq B$ be an extension of rings. Show that the integral closure of A in B is integrally closed in B.
(c) Let $A \subseteq B$ be an integral extension of rings, $\mathfrak{b}$ an ideal of B and $\mathfrak{a} := A \cap \mathfrak{b}$. (Note that $\mathfrak{a}$ is an ideal of A. If $\mathfrak{b}$ is prime in B, then $\mathfrak{a}$ is prime in A. See Proposition 2.10.) Show that $B/\mathfrak{b}$ is integral over $A/\mathfrak{a}$.

2.127 Let $A \subseteq B$ be an extension of integral domains, $\mathfrak{a}$ a finitely generated non-zero ideal of A and $\gamma \in B$. If $\gamma\mathfrak{a} \subseteq \mathfrak{a}$, show that γ is integral over A. **[H]**

2.128 **(a)** Let $A \subseteq B$ be an integral extension of integral domains. Show that A is a field if and only if B is a field.
(b) Let $A \subseteq B$ be an integral extension of rings, $\mathfrak{q}$ a prime ideal of B and $\mathfrak{p} := A \cap \mathfrak{q}$. Show that $\mathfrak{p}$ is maximal if and only if $\mathfrak{q}$ is maximal. **[H]**
(c) Let A, B, $\mathfrak{p}$ and $\mathfrak{q}$ be as in (b). Further let $\mathfrak{q}'$ be another prime ideal of B with $\mathfrak{p} = A \cap \mathfrak{q}'$. Show that if $\mathfrak{q} \subseteq \mathfrak{q}'$, then $\mathfrak{q} = \mathfrak{q}'$. **[H]**

2.129 Let A be a ring and S a multiplicatively closed subset of A. Show that:
(a) If $0 \in S$, then $S^{-1}A$ is the zero ring.
(b) If $S' := S \setminus \{1\}$ is non-empty and closed under multiplication, then $S'^{-1}A \cong S^{-1}A$.
(c) If A is Noetherian, then $S^{-1}A$ is also Noetherian.

2.130 Let $A \subseteq B$ be a ring extension and C the integral closure of A in B. Show that for any multiplicative subset S of A (and hence of B and C) the integral closure of $S^{-1}A$ in $S^{-1}B$ is $S^{-1}C$. In particular, if A is integrally closed in B, then so is $S^{-1}A$ in $S^{-1}B$.

2.131 Recall that an integrally closed integral domain is called a *normal domain* (ND).
(a) Show that every UFD is a normal domain.
(b) Let D be a square-free integer $\neq 0, 1$. Show that $\mathbb{Z}[\sqrt{D}] = \{a + b\sqrt{D} \mid a, b \in \mathbb{Z}\}$ is normal if and only if $D \equiv 2, 3 \pmod 4$.
(**Remark:** The reader should note the following important implications:

$$\boxed{\text{ED} \Rightarrow \text{PID} \Rightarrow \text{UFD} \Rightarrow \text{ND}}$$

That is, a Euclidean domain is a PID, a PID is a UFD and a UFD is a normal domain. Neither of the reverse implications is true. For example, the ring $\mathbb{Z}[\frac{1+\sqrt{-19}}{2}]$ of integers of $\mathbb{Q}(\sqrt{-19})$ is known to be a PID but not a Euclidean domain. The ring $K[X_1, \ldots, X_n]$, $n \geqslant 2$, of multivariate polynomials over a field K is a UFD, but not a PID, since the ideal $\langle X_1, \ldots, X_n \rangle$ is not principal. Finally, $\mathbb{Z}[\sqrt{-5}]$ is a normal domain (by Exercise 2.136 below), but not a UFD, since $6 = 2 \cdot 3 = (1 + \sqrt{-5})(1 - \sqrt{-5})$ are two different factorizations of 6 into irreducible elements.)

2.132 A (non-zero) ring A with a unique maximal ideal $\mathfrak{m}$ is called a *local ring*. In that case, the field $A/\mathfrak{m}$ is called the *residue field* of A.
Let A be ring and $\mathfrak{p}$ a prime ideal of A. Show that the localization $A_\mathfrak{p}$ is a local ring with the unique maximal ideal $\mathfrak{p}A_\mathfrak{p}$ generated by elements a/b, $a \in \mathfrak{p}$, $b \notin \mathfrak{p}$, and the residue field $A_\mathfrak{p}/\mathfrak{p}A_\mathfrak{p}$ is canonically isomorphic to the quotient field $\mathrm{Q}(A/\mathfrak{p})$ of the integral domain $A/\mathfrak{p}$ under the map $(a/b) + \mathfrak{p}A_\mathfrak{p} \mapsto (a+\mathfrak{p})/(b+\mathfrak{p})$, $a \in A$, $b \in A \setminus \mathfrak{p}$.

2.133 A ring A is called a *discrete valuation ring* (DVR) or a *discrete valuation domain* (DVD), if A is a local principal ideal domain. Let A be a DVR with maximal ideal $\mathfrak{m} = \langle p \rangle$. Prove the following assertions:
(a) A is a UFD.
(b) The only primes in A are the associates of p. **[H]**
(c) Every non-zero element of A can be written as up^α, where u is a unit of A and $\alpha \in \mathbb{N}_0$.
(d) Every non-zero ideal of A is of the form $\langle p^\alpha \rangle$ for some $\alpha \in \mathbb{N}_0$.
(e) A has only one non-zero prime ideal (namely, $\mathfrak{m}$).
(**Remark:** The prime p of A is called a *uniformizing parameter* or a *uniformizer* for A and is unique up to multiplication by units.
The map $\nu : A \setminus \{0\} \to \mathbb{N}_0$ taking $up^\alpha \mapsto \alpha$ is called a *discrete valuation* of A and can be naturally extended to a group homomorphism $\nu : K^* \to \mathbb{Z}$ by defining $\nu(a/b) := \nu(a) - \nu(b)$, where $a, b \in A$, $b \neq 0$ and $K = \mathrm{Q}(A)$ is the quotient field of A. It is often convenient to define $\nu(0) := +\infty$. It follows that $A = \{x \in K \mid \nu(x) \geqslant 0\}$ and $\mathfrak{m} = \{x \in K \mid \nu(x) > 0\}$.)

2.134 **(a)** Let A be a local Noetherian integral domain which is not a field. Assume further that the maximal ideal $\mathfrak{m} \neq 0$ of A is the only non-zero prime ideal of A. Show that A is a DVR (that is, a PID) if and only if A is integrally closed.
(b) Let A be a Noetherian integral domain which is not a field. Prove that A is a Dedekind domain if and only if $A_\mathfrak{p}$ is a DVR for every non-zero prime ideal $\mathfrak{p}$ of A.

2.135 **(a)** Show that the only units of $\mathbb{Z}[\mathrm{i}]$ are ± 1 and $\pm \mathrm{i}$.
(b) Show that the primes of $\mathbb{Z}[\mathrm{i}]$ are associates to the following:
(1) a prime integer $\equiv 3 \pmod 4$,
(2) $a + \mathrm{i}b$, $a, b \in \mathbb{Z}$, with $a^2 + b^2$ equal to 2 or a prime integer $\equiv 1 \pmod 4$.

2.136 **(a)** Show that every quadratic number field K can be represented as $K \cong \mathbb{Q}(\sqrt{D})$ for a *square-free* integer $D \neq 0, 1$.
(b) Let $K = \mathbb{Q}(\sqrt{D})$ for some square-free integer $D \neq 0, 1$. Show that:

$$\mathfrak{O}_K \cong \begin{cases} \mathbb{Z}[\frac{1+\sqrt{D}}{2}] \cong \mathbb{Z}[X]/\langle X^2 - X + \frac{1-D}{4} \rangle & \text{if } D \equiv 1 \pmod 4, \\ \mathbb{Z}[\sqrt{D}] \cong \mathbb{Z}[X]/\langle X^2 - D \rangle & \text{if } D \equiv 2, 3 \pmod 4. \end{cases}$$

(In particular, the ring of integers of $\mathbb{Q}(\mathrm{i}) = \{a + \mathrm{i}b \mid a, b \in \mathbb{Q}\}$ is the ring $\mathbb{Z}[\mathrm{i}] = \{a + \mathrm{i}b \mid a, b \in \mathbb{Z}\}$ of *Gaussian integers*.)

2.137 Let A be a Dedekind domain.
(a) Let $\mathfrak{q}_1$ and $\mathfrak{q}_2$ be two distinct non-zero prime ideals of A. Show that for any $e_1, e_2 \in \mathbb{N}$ we have $\mathfrak{q}_1^{e_1} + \mathfrak{q}_2^{e_2} = A$. **[H]**
(b) Let $\mathfrak{a} = \mathfrak{q}_1^{e_1} \cdots \mathfrak{q}_r^{e_r}$ be the prime factorization of a non-zero ideal $\mathfrak{a}$ of A with pairwise distinct primes $\mathfrak{q}_i$ and $e_i \in \mathbb{N}$. Show that $A/\mathfrak{a} \cong \prod_{i=1}^r (A/\mathfrak{q}_i^{e_i})$. **[H]**

2.138 Let A be a Dedekind domain and $\mathfrak{a}$ a non-zero (integral) ideal of A. Show that:
(a) There exists a non-zero (integral) ideal $\mathfrak{b}$ of A such that $\mathfrak{ab}$ is a principal ideal. **[H]**
(b) The number of ideals of A containing $\mathfrak{a}$ is finite.

(c) Every ideal of $A/\mathfrak{a}$ is principal.

2.139 Let $\mathfrak{a} := \mathfrak{q}_1^{e_1} \cdots \mathfrak{q}_r^{e_r}$ and $\mathfrak{b} := \mathfrak{q}_1^{f_1} \cdots \mathfrak{q}_r^{f_r}$, $e_i, f_i \in \mathbb{N}_0$, be the prime decompositions of two non-zero ideals $\mathfrak{a}, \mathfrak{b}$ of a DD A. Define the *gcd* and *lcm* of $\mathfrak{a}$ and $\mathfrak{b}$ as

$$\begin{aligned} \gcd(\mathfrak{a}, \mathfrak{b}) &:= \mathfrak{q}_1^{\min(e_1,f_1)} \cdots \mathfrak{q}_r^{\min(e_r,f_r)}, \\ \operatorname{lcm}(\mathfrak{a}, \mathfrak{b}) &:= \mathfrak{q}_1^{\max(e_1,f_1)} \cdots \mathfrak{q}_r^{\max(e_r,f_r)}. \end{aligned}$$

Show that $\gcd(\mathfrak{a}, \mathfrak{b}) = \mathfrak{a} + \mathfrak{b}$ and $\operatorname{lcm}(\mathfrak{a}, \mathfrak{b}) = \mathfrak{a} \cap \mathfrak{b}$. Conclude that $\gcd(\mathfrak{a}, \mathfrak{b}) \operatorname{lcm}(\mathfrak{a}, \mathfrak{b}) = (\mathfrak{a} + \mathfrak{b})(\mathfrak{a} \cap \mathfrak{b}) = \mathfrak{a}\mathfrak{b}$. (Note that if A is a general ring, we only have $(\mathfrak{a} + \mathfrak{b})(\mathfrak{a} \cap \mathfrak{b}) = \mathfrak{a}(\mathfrak{a} \cap \mathfrak{b}) + \mathfrak{b}(\mathfrak{a} \cap \mathfrak{b}) \subseteq \mathfrak{a}\mathfrak{b}$.)

2.140 Let K be a number field and $\mathfrak{O} := \mathfrak{O}_K$.
(a) Let $\mathfrak{a}$ be an ideal of $\mathfrak{O}$. Show that $\mathrm{N}(\mathfrak{a}) \in \mathfrak{a}$. In particular, every non-zero ideal of $\mathfrak{O}$ contains a non-zero integer. **[H]**
(b) Let $\mathfrak{q}$ be a non-zero prime ideal of $\mathfrak{O}$. Prove that $\mathrm{N}(\mathfrak{q}) = p^f$ for some $f \in \mathbb{N}$, where p is the unique rational prime contained in $\mathfrak{q}$ (Lemma 2.13).

2.141 Let K be a number field, $\mathfrak{O} := \mathfrak{O}_K$, $\mathfrak{U} := \mathfrak{U}_K$, $\mathfrak{R} := \mathfrak{R}_K$ and $\alpha \in \mathfrak{O}$. Show that:
(a) $\alpha \in \mathfrak{U}$, if and only if $\mathrm{N}(\alpha) = \pm 1$.
(b) $\alpha \in \mathfrak{U}$, if and only if $f(0) = \pm 1$, where $f(X) \in \mathbb{Q}[X]$ is the minimal polynomial of α over $\mathbb{Q}$.
(c) $\alpha \in \mathfrak{R}$, if and only if $|\sigma(\alpha)| = 1$ for every complex embedding σ of K.

2.142 Let K be a number field. We say that K is *norm-Euclidean*, if for every $\alpha, \beta \in \mathfrak{O}_K$, $\beta \neq 0$, there exist $q, r \in \mathfrak{O}_K$ such that $\alpha = q\beta + r$ and $|\mathrm{N}(r)| < |\mathrm{N}(\beta)|$.
(a) Conclude that if K is norm-Euclidean, then $\mathfrak{O}_K$ is a Euclidean domain with the Euclidean degree function $\nu(\alpha) := |\mathrm{N}(\alpha)|$. (The converse of this is not true. For example, it is known that $K := \mathbb{Q}(\sqrt{69})$ is not norm-Euclidean, but $\mathfrak{O}_K$ is a Euclidean domain.)
(b) Prove the following equivalent characterization of a norm-Euclidean number field: K is norm-Euclidean if and only if for every $\alpha \in K$ there exists $\beta \in \mathfrak{O}_K$ such that $|\mathrm{N}(\alpha - \beta)| < 1$.
(c) Show that the following number fields are norm-Euclidean:
$\mathbb{Q}$, $\mathbb{Q}(\sqrt{-1})$, $\mathbb{Q}(\sqrt{-2})$, $\mathbb{Q}(\sqrt{2})$ and $\mathbb{Q}(\sqrt{3})$.
(d) Show that $\mathbb{Q}(\sqrt{-6})$ is not norm-Euclidean. **[H]**

2.143 In this exercise, one derives that the only (rational) integer solutions of *Bachet's equation*

$$y^2 = x^3 - 2 \tag{2.20}$$

are $x = 3$, $y = \pm 5$.
(a) Show that Equation (2.20) has no solutions with x or y even. **[H]**
Let (x, y) be a solution of Equation (2.20) with both x and y odd. Then x^3 admits a factorization in $\mathbb{Z}[\sqrt{-2}]$ as $x^3 = (y + \sqrt{-2})(y - \sqrt{-2})$.
(b) Let $K := \mathbb{Q}(\sqrt{-2})$. Show that $\mathfrak{O}_K = \mathbb{Z}[\sqrt{-2}]$ and that $\mathfrak{O}_K$ is a UFD. Also the only units of $\mathfrak{O}_K$ are ± 1.

(c) Show that $\gcd(y+\sqrt{-2}, y-\sqrt{-2}) = 1$. **[H]**
(d) Because of unique factorization one can write $y+\sqrt{-2} = \pm(c+d\sqrt{-2})^3$ for $c, d \in \mathbb{Z}$. Expand the cube and equate the real and imaginary parts to conclude that we must have $y = \pm 5$, so that $x = 3$.

** 2.14 p-adic Numbers

Let us now study a different area of algebraic number theory, introduced by Kurt Hensel in an attempt to apply power series expansions in connection with numbers. While trying to explain the properties of (rational) integers mathematicians started embedding $\mathbb{Z}$ in bigger and bigger structures, richer and richer in properties. $\mathbb{Q}$ came in a natural attempt to form quotients, and for some time people believed that that is all about *real*ity. Pythagoras was seemingly the first to locate and prove the irrationality of a number, namely, $\sqrt{2}$. It took humankind centuries for completing the picture of the *real line*. One possibility is to look $\mathbb{R}$ as the *completion* of $\mathbb{Q}$. A sequence a_n, $n \in \mathbb{N}$, of rational numbers is called a *Cauchy sequence* if for every real $\epsilon > 0$, there exists $N \in \mathbb{N}$ such that $|a_m - a_n| \leqslant \epsilon$ for all $m, n \in \mathbb{N}$, $m, n \geqslant N$. Every Cauchy sequence should converge to a limit and it is $\mathbb{R}$ (and not $\mathbb{Q}$) where this happens. Seeing convergence of Cauchy sequences, people were not wholeheartedly happy, because the real polynomial X^2+1 did not have—it continues not to have—roots in $\mathbb{R}$. So the next question that arose was that of *algebraic closure*. $\mathbb{C}$ was invented and turned out to be a nice field which is both algebraically closed and complete.

Throughout the above business, we were led by the conventional notion of *distance* between points (that is, between numbers)—the so-called *Archimedean distance* or the *absolute value*. For every rational prime p, there exists a *p-adic distance* which leads to a ring $\hat{\mathbb{Z}}_p$ strictly bigger than and containing $\mathbb{Z}$. This is the ring of *p-adic integers*. The quotient field of $\hat{\mathbb{Z}}_p$ is the field $\mathbb{Q}_p$ of p-adic numbers. $\mathbb{Q}_p$ is complete in the sense of convergence of Cauchy sequences (under the p-adic distance), but is not algebraically closed. We know anyway that a (unique) algebraic closure $\bar{\mathbb{Q}}_p$ of $\mathbb{Q}_p$ exists. We have $[\mathbb{C} : \mathbb{R}] = 2$, that is, it was necessary and sufficient to add the imaginary quantity i to $\mathbb{R}$ to get an algebraically closed field. Unfortunately in the case of the p-adic distance the closure $\bar{\mathbb{Q}}_p$ is of infinite extension degree over $\mathbb{Q}_p$. In addition, $\bar{\mathbb{Q}}_p$ is not complete. An attempt to make $\bar{\mathbb{Q}}_p$ complete gives an even bigger field Ω_p and the story stops here, Ω_p being both algebraically closed and complete. But Ω_p is already a pretty huge field and very little is known about it.

In the rest of this section, we, without specific mention, denote by p an arbitrary rational prime.

2.14.1 The Arithmetic of p-adic Numbers

There are various ways in which p-adic integers can be defined. A simple way is to use infinite sequences.

Definition 2.111 A *p-adic integer* is defined as an infinite sequence $(a_n) = (a_n)_{n\in\mathbb{N}}$ of elements $a_n \in \mathbb{Z}_{p^n} = \mathbb{Z}/p^n\mathbb{Z}$ with the property that $a_{n+1} \equiv a_n \pmod{p^n}$ for every $n \in \mathbb{N}$. Each

a_n, being an element of $\mathbb{Z}_{p^n}$, can be represented as a (rational) integer unique modulo p^n. Thus, if b_n, $n \in \mathbb{N}$, define another sequence of integers with $b_n \equiv a_n \pmod{p^n}$ for every n, the p-adic integers (a_n) and (b_n) are treated the same. In particular, if $0 \leqslant b_n < p^n$ for every n, then (b_n) is called the *canonical representation* of (a_n). The set of all p-adic integers is denoted by $\hat{\mathbb{Z}}_p$.[20] A sequence (a_n) of integers with $a_{n+1} \equiv a_n \pmod{p^n}$ for every n is called a *p-coherent sequence*. □

See Exercise 2.144 for another way of defining p-adic integers. We now show that $\hat{\mathbb{Z}}_p$ is a ring. Before doing that, we mention that the ring $\mathbb{Z}$ is canonically embedded in $\hat{\mathbb{Z}}_p$ by the injective map $\iota : \mathbb{Z} \to \hat{\mathbb{Z}}_p$, $a \mapsto (a)$.

Definition 2.112 Let (a_n) and (b_n) be two p-adic integers. Define:

$$\begin{aligned}(a_n) + (b_n) &:= (a_n + b_n).\\ (a_n) \cdot (b_n) &:= (a_n \cdot b_n).\end{aligned}$$

One can easily check that these operations are well-defined, that is, independent of the choice of the representatives of a_n and b_n. It also follows easily that these operations make $\hat{\mathbb{Z}}_p$ a ring with additive identity $\iota(0) = (0)$ and with multiplicative identity $\iota(1) = (1)$. The additive inverse of (a_n) is $-(a_n) = (-a_n)$. Moreover ι is an injective ring homomorphism $\mathbb{Z} \to \hat{\mathbb{Z}}_p$. In view of this, one often identifies the rational integer a with the p-adic integer $\iota(a) = (a)$. We will also do so, provided that we do not expect to face a danger of confusion. Also note that for $l \in \mathbb{Z}$ the l-fold sum $l(a_n)$ is the same as $(l)(a_n) = (la_n)$. Thus in this context the two interpretations of l remain perfectly consistent. □

It turns out that $\hat{\mathbb{Z}}_p$ is an integral domain. In order to see why, let us focus our attention on the units of $\hat{\mathbb{Z}}_p$. Let us plan to denote $\hat{\mathbb{Z}}_p^*$ (the multiplicative group of units of $\hat{\mathbb{Z}}_p$) by U_p. The next result characterizes elements of U_p.

Proposition 2.52 For $(a_n) \in \hat{\mathbb{Z}}_p$, the following conditions are equivalent:

(a) $(a_n) \in U_p$
(b) $p \not| a_n$ for all $n \in \mathbb{N}$.
(c) $p \not| a_1$.

Proof [(a)⇒(b)] Let $(a_n)(b_n) = (a_n b_n) = 1 = (1)$ for some $(b_n) \in \hat{\mathbb{Z}}_p$. Then for every $n \in \mathbb{N}$ we have $a_n b_n \equiv 1 \pmod{p^n}$, that is, a_n is invertible modulo p^n and hence modulo p as well, that is, $p \not| a_n$.
[(b)⇒(c)] Obvious.
[(c)⇒(a)] Let us construct a p-coherent sequence b_n, $n \in \mathbb{N}$, of (rational) integers with $a_n b_n \equiv 1 \pmod{p^n}$. This (b_n) would be the desired inverse of (a_n) in $\hat{\mathbb{Z}}_p$. Since $p \not| a_1$ and $a_n \equiv a_1 \pmod{p}$, it follows that $p \not| a_n$ as well and, therefore, the congruence $a_n x \equiv 1 \pmod{p^n}$ has a unique solution modulo p^n, namely $b_n \equiv a_n^{-1} \pmod{p^n}$.

[20] Well! We are now in a mess of notations. We have $\mathbb{Z}_n := \mathbb{Z}/n\mathbb{Z}$ for every $n \in \mathbb{N}$. In particular, for $p \in \mathbb{P}$ we have $\mathbb{Z}_p$ which is a field that we planned to denote also by $\mathbb{F}_p$. It is superfluous to have two notations for the same thing. Many authors, therefore, prefer to avoid the hat and call $\hat{\mathbb{Z}}_p$ as $\mathbb{Z}_p$. For them, our $\mathbb{Z}_p$ is $\mathbb{F}_p$ and/or $\mathbb{Z}/p\mathbb{Z}$ written explicitly. Let us stick to our old conventions and use hats to remove ambiguities.

We also have $a_{n+1}b_{n+1} \equiv 1 \pmod{p^n}$, that is, $a_n b_{n+1} \equiv 1 \pmod{p^n}$, that is, $b_{n+1} \equiv a_n^{-1} \equiv b_n \pmod{p^n}$. ■

Proposition 2.53 Every $0 \neq x = (a_n) \in \hat{\mathbb{Z}}_p$ can be written uniquely as $x = p^r y$ for some $r \in \mathbb{N}_0$ and for some $y \in U_p$.

Proof If $p \nmid a_1$, take $r := 0$ and $y := x$. So assume that $p|a_1$. Choose $r \in \mathbb{N}$ such that $[a_n]_{p^n} = [0]_{p^n}$ for $1 \leqslant n \leqslant r$, whereas $[a_{r+1}]_{p^{r+1}} \neq [0]_{p^{r+1}}$. Such an r exists, since $x \neq 0$ by hypothesis. For $n \in \mathbb{N}$, we have $a_{r+n} \equiv a_r \equiv 0 \pmod{p^r}$, that is, $p^r | a_{r+n}$, whereas $a_{r+n} \equiv a_{r+1} \not\equiv 0 \pmod{p^{r+1}}$, that is, $p^{r+1} \nmid a_{r+n}$, that is, $\mathrm{v}_p(a_{r+n}) = r$. Define $b_n := a_{r+n}/p^r$. Since $a_{r+n+1} \equiv a_{r+n} \pmod{p^{r+n}}$, division by p^r gives $b_{n+1} \equiv b_n \pmod{p^n}$, that is, $y := (b_n) \in \hat{\mathbb{Z}}_p$. Moreover, $p^r b_n = a_{r+n} \equiv a_n \pmod{p^n}$, that is, $x = p^r y$. Finally, since $p \nmid b_1$, we have $y \in U_p$. This establishes the existence of a factorization $x = p^r y$. The uniqueness of this factorization is left to the reader as an easy exercise. ■

Proposition 2.54 $\hat{\mathbb{Z}}_p$ is an integral domain.

Proof Let x_1 and x_2 be non-zero elements of $\hat{\mathbb{Z}}_p$. By Proposition 2.53, we can write $x_1 = p^{r_1} y_1$ and $x_2 = p^{r_2} y_2$ with $r_1, r_2 \in \mathbb{N}_0$ and $y_1, y_2 \in U_p$. Then $(a_n) := x_1 x_2 = p^{r_1+r_2} y_1 y_2$. Now $(b_n) := y_1 y_2 \in U_p$ and hence no b_n is divisible by p. Therefore, $a_{r_1+r_2+1} = p^{r_1+r_2} b_{r_1+r_2+1} \not\equiv 0 \pmod{p^{r_1+r_2+1}}$, that is, $(a_n) = x_1 x_2 \neq 0$. ■

Definition 2.113 The quotient field $\mathbb{Q}_p := \mathrm{Q}(\hat{\mathbb{Z}}_p)$ of $\hat{\mathbb{Z}}_p$ is called the *field of p-adic numbers.* □

Proposition 2.55 Every non-zero $x \in \mathbb{Q}_p$ can be expressed uniquely as $x = p^r y$ with $r \in \mathbb{Z}$ and $y \in U_p$.

Proof One can write $x = a/b$ for some $a, b \in \hat{\mathbb{Z}}_p \setminus \{0\}$. Then $a = p^s c$ and $b = p^t d$ for some $s, t \in \mathbb{N}_0$, $c, d \in U_p$ and so $x = p^{s-t}(c/d)$ with $c/d = cd^{-1} \in U_p$. The proof for the uniqueness is left to the reader. ■

The canonical inclusion $\iota : \mathbb{Z} \hookrightarrow \hat{\mathbb{Z}}_p$ naturally extends to the canonical inclusion $\iota : \mathbb{Q} \hookrightarrow \mathbb{Q}_p$. We can identify $\iota(a/b) = \iota(a)/\iota(b) \in \iota(\mathbb{Q}) \subseteq \mathbb{Q}_p$ with the rational a/b and say that $\mathbb{Q}$ is *contained* in $\mathbb{Q}_p$. Being a field of characteristic 0, $\mathbb{Q}_p$ contains an isomorphic copy of $\mathbb{Q}$. The map ι gives this isomorphism explicitly. Note that the ring $\hat{\mathbb{Z}}_p$ is strictly bigger than $\mathbb{Z}$ and the field $\mathbb{Q}_p$ is strictly bigger than the field $\mathbb{Q}$ (Exercise 2.147).

2.14.2 The p-adic Valuation

Proposition 2.55 leads to the notion of p-adic distance between pairs of points in $\mathbb{Q}_p$. Let us start with some formal definitions.

Definition 2.114 A *metric* on a set S is a map $d : S \times S \to \mathbb{R}$ such that for every $x, y, z \in S$ we have:

(1) **Non-negative** $d(x, y) \geqslant 0$.

(2) **Non-degeneracy** $d(x, y) = 0$ if and only if $x = y$.

(3) **Symmetry** $d(x, y) = d(y, x)$.

(4) **Triangle inequality** $d(x,z) \leqslant d(x,y) + d(y,z)$.

A set S together with a metric d is called a *metric space* (with metric d). □

Definition 2.115 A *norm* on a field K is a map $||\ ||: K \to \mathbb{R}$ such that for all $x, y \in K$ we have:

(1) **Non-negative** $||x|| \geqslant 0$.
(2) **Non-degeneracy** $||x|| = 0$ if and only if $x = 0$.
(3) **Multiplicativity** $||xy|| = ||x||\,||y||$.
(4) **Triangle inequality** $||x+y|| \leqslant ||x|| + ||y||$.

It is an easy check that for a norm $||\ ||$ on K the function $d: K \times K \to \mathbb{R}$, $d(x,y) := ||x-y||$, defines a metric on K.

A norm $||\ ||$ on a field K is called *non-Archimedean* (or a *finite valuation*), if $||x+y|| \leqslant \max(||x||, ||y||)$ for all $x, y \in K$ (a condition stronger than the triangle inequality). A norm which is not non-Archimedean is called *Archimedean* (or an *infinite valuation*). □

Example 2.34

(1) Setting $||x|| := \begin{cases} 0 & \text{if } x = 0, \\ 1 & \text{if } x \neq 0, \end{cases}$ defines a norm on any field K. This norm is called the *trivial norm* on K.

(2) The absolute value $|\ |$ is an *Archimedean norm* on $\mathbb{Q}$ (or $\mathbb{R}$). It is customary to denote this norm as $|\ |_\infty$. This norm induces the usual metric topology on $\mathbb{Q}$ (or $\mathbb{R}$) which is at the heart of *real analysis*. In *p-adic analysis*, one investigates $\mathbb{Q}$ under the p-adic norms that we define now. □

Definition 2.116 The *p-adic norm* $|\ |_p: \mathbb{Q}_p \to \mathbb{R}$ on $\mathbb{Q}_p$ is defined as:

$$|x|_p := \begin{cases} 0 & \text{if } x = 0, \\ p^{-r} & \text{if } x = p^r y \text{ with } r \in \mathbb{Z} \text{ and } y \in U_p. \end{cases}$$ □

Theorem 2.59 The p-adic norm $|\ |_p$ is a non-Archimedean norm on $\mathbb{Q}_p$.

Proof Non-negative-ness, non-degeneracy and multiplicativity of $|\ |_p$ are immediate. For proving the triangle inequality, it is sufficient to prove the non-Archimedean condition. Take $x, y \in \mathbb{Q}_p$. If $x = 0$ or $y = 0$ or $x + y = 0$, we clearly have $|x+y|_p \leqslant \max(|x|_p, |y|_p)$. So assume that each of x, y and $x+y$ is non-zero. Write $x = p^r u$ and $y = p^s v$ with $r, s \in \mathbb{Z}$ and $u, v \in U_p$. Without loss of generality, we may assume that $r \geqslant s$. Then, $x + y = p^s z$, where $z := p^{r-s}u + v \in \hat{\mathbb{Z}}_p$. Since $x + y \neq 0$, we have $z \neq 0$; so we can write $z = p^t w$ for some $t \in \mathbb{N}_0$ and $w \in U_p$. But then $|x+y|_p = p^{-(s+t)} \leqslant p^{-s} = \max(p^{-r}, p^{-s}) = \max(|x|_p, |y|_p)$. ■

Definition 2.117 Two metrics d_1 and d_2 on a metric space S are called *equivalent* if a sequence (x_n) from S is Cauchy with respect to d_1 if and only if it is Cauchy with respect to d_2. Two norms on a field are called equivalent if they induce equivalent metrics. □

For every $p \in \mathbb{P}$, the field $\mathbb{Q}$ is canonically embedded in $\mathbb{Q}_p$ and thus we have a notion of a p-adic *distance* on $\mathbb{Q}$. We also have the usual Archimedean distance $|\ |_\infty$

on $\mathbb{Q}$. We now state an interesting result without a proof, which asserts that any distance on $\mathbb{Q}$ must be essentially the same as either the usual Archimedean distance or one of the p-adic distances.

Theorem 2.60 **Ostrowski's theorem** Every non-trivial norm on $\mathbb{Q}$ is equivalent to $|\ |_p$ for some $p \in \mathbb{P} \cup \{\infty\}$. ■

The notions of sequences and series and their convergences can be readily extended to $\mathbb{Q}_p$ under the norm $|\ |_p$. Since the p-adic distance assumes only the discrete values p^r, $r \in \mathbb{Z}$, it is often customary to restrict ourselves only to these values while talking about the convergence criteria of sequences and series, that is, instead of an infinitesimally small real $\epsilon > 0$ one can talk about an arbitrarily large $M \in \mathbb{N}$ with $p^{-M} \leqslant \epsilon$.

Definition 2.118 Let $x_1, x_2, \ldots$ be a sequence of elements of $\mathbb{Q}_p$. We say that this sequence *converges* to a *limit* $x \in \mathbb{Q}_p$, if given $M \in \mathbb{N}$ there exists $N \in \mathbb{N}$ such that $|x_n - x|_p \leqslant p^{-M}$ for all $n \geqslant N$. We write this as $x = \lim x_n$ or as $x_n \to x$.

Consider the partial sums $s_n := x_1 + \cdots + x_n \in \mathbb{Q}_p$ for each $n \in \mathbb{N}$. If there exists $s \in \mathbb{Q}_p$ with $s_n \to s$, we say that the *sum* $\sum_{n\in\mathbb{N}} x_n$ *converges* to s and write $s = \sum_{n\in\mathbb{N}} x_n$.

A sequence $x_1, x_2, \ldots$ of elements of $\mathbb{Q}_p$ is said to be a *Cauchy sequence* if for every $M \in \mathbb{N}$, there exists an $N \in \mathbb{N}$ such that $|x_m - x_n|_p \leqslant p^{-M}$ for all $m, n \geqslant N$. □

Definition 2.119 A field K is called *complete* under a norm $||\ ||$ if every sequence of elements of K, which is Cauchy under $||\ ||$, converges to an element in K. □

For example, $\mathbb{R}$ is complete under $|\ |_\infty$. We shortly demonstrate that $\mathbb{Q}_p$ is complete under $|\ |_p$.

Consider a field K not (necessarily) complete under a norm $||\ ||$. Let C denote the set of all Cauchy sequences $(a_n) = (a_n)_{n\in\mathbb{N}}$ from K. Define addition and multiplication in C as $(a_n) + (b_n) := (a_n + b_n)$ and $(a_n)(b_n) := (a_n b_n)$. Under these operations C becomes a commutative ring with identity having a maximal ideal $\mathfrak{m} := \{(a_n) \mid a_n \to 0\}$. The field $L := C/\mathfrak{m}$ is called the *completion* of K with respect to the norm $||\ ||$. K is canonically embedded in L via the map $x \mapsto (x) + \mathfrak{m}$. The norm $||\ ||$ on K extends to elements $(a_n) + \mathfrak{m}$ of L as $\lim_{n\to\infty} ||a_n||$. L is a complete field under this extended norm. In fact, it is the smallest field containing K and complete under $||\ ||$.

$\mathbb{R}$ is the *completion* of $\mathbb{Q}$ with respect to the Archimedean norm $|\ |_\infty$. On the other hand, $\mathbb{Q}_p$ turns out to be the completion of $\mathbb{Q}$ with respect to the p-adic norm $|\ |_p$. Before proving this let us first prove that $\mathbb{Q}_p$ itself is a complete field under the p-adic norm. Let us start with a lemma.

Lemma 2.18 A sequence (a_n) of p-adic numbers is a Cauchy sequence if and only if the sequence $(a_{n+1} - a_n)$ converges to 0.

Proof [if] Take any $M \in \mathbb{N}$. Since $a_{n+1} - a_n \to 0$ by hypothesis, there exists $N \in \mathbb{N}$ such that $|a_{n+1} - a_n|_p \leqslant p^{-M}$ for all $n \geqslant N$. But then for all $m, n \geqslant N$ with $m = n+k$, $k \in \mathbb{N}$, we have $|a_m - a_n|_p = \left|\sum_{i=0}^{k-1}(a_{n+i+1} - a_{n+i})\right|_p \leqslant \max_{0\leqslant i\leqslant k-1} |a_{n+i+1} - a_{n+i}|_p \leqslant p^{-M}$. Thus (a_n) is a Cauchy sequence.

[only if] Take any $M \in \mathbb{N}$. Since (a_n) is a Cauchy sequence by hypothesis, there exists $N \in \mathbb{N}$ such that $|a_m - a_n|_p \leqslant p^{-M}$ for all $m, n \geqslant N$. In particular, $|a_{n+1} - a_n|_p \leqslant p^{-M}$ for all $n \geqslant N$, that is, $a_{n+1} - a_n \to 0$. ■

Theorem 2.61 The field $\mathbb{Q}_p$ is complete with respect to $|\ |_p$.

Proof Let (a_n) be a Cauchy sequence in $\mathbb{Q}_p$. By Lemma 2.18, $a_{n+1} - a_n \to 0$. Therefore, there exists $N \in \mathbb{N}$ such that $|a_{n+1} - a_n|_p \leqslant 1$ for all $n \geqslant N$. For $n = N + k$, $k \in \mathbb{N}$, we have

$$\begin{aligned} |a_n|_p &= |a_{N+k}|_p \\ &= |(a_{N+k} - a_{N+k-1}) + \cdots + (a_{N+1} - a_N) + a_N|_p \\ &\leqslant \max(|a_{N+k} - a_{N+k-1}|_p, \ldots, |a_{N+1} - a_N|_p, |a_N|_p) \\ &\leqslant \max(1, |a_N|_p). \end{aligned}$$

It then follows that $|a_n|_p \leqslant p^{-m}$ for all $n \in \mathbb{N}$, where $m \in \mathbb{Z}$ satisfies $p^{-m} = \max(1, |a_1|_p, \ldots, |a_N|_p)$. If $m \geqslant 0$, then each $a_n \in \hat{\mathbb{Z}}_p$ (Exercise 2.148). Otherwise consider the sequence $(p^{-m}a_n)$ which is clearly Cauchy and in which each $p^{-m}a_n \in \hat{\mathbb{Z}}_p$, since $|p^{-m}a_n|_p \leqslant p^m p^{-m} = 1$. Thus, without loss of generality, we may assume that the given sequence (a_n) itself is one of p-adic *integers*.

Let $a_n = a_{n,0} + a_{n,1}p + a_{n,2}p^2 + \cdots$ be the p-adic expansion of a_n (Exercise 2.145). Since (a_n) is Cauchy, for every $M \in \mathbb{N}_0$ there exists $N_M \in \mathbb{N}$ such that $|a_m - a_n|_p \leqslant p^{-(M+1)}$ for all $m, n \geqslant N_M$: that is, $a_{n,i} = a_{m,i}$ for $0 \leqslant i \leqslant M$, $m, n \geqslant N_M$. Define $x_M := a_{n,M}$ for any $n \geqslant N_M$ and $x := x_0 + x_1 p + x_2 p^2 + \cdots \in \hat{\mathbb{Z}}_p$. It then follows that $a_n \to x$. ■

Theorem 2.62 $\mathbb{Q}_p$ is the completion of $\mathbb{Q}$ with respect to the norm $|\ |_p$.

Proof Let C denote the ring of Cauchy sequences from $\mathbb{Q}$ (under the p-adic norm), $\mathfrak{m}$ the maximal ideal of C consisting of sequences that converge to 0, and $L := C/\mathfrak{m}$. We now show that $L \cong \mathbb{Q}_p$.

If $a \in \mathbb{Q}_p$ has the p-adic expansion $a = a_{-r}p^{-r} + \cdots + a_{-1}p^{-1} + a_0 + a_1 p + a_2 p^2 + \cdots$ (Exercise 2.145), then $\alpha_n := a_{-r}p^{-r} + \cdots + a_{-1}p^{-1} + a_0 + a_1 p + \cdots + a_n p^n$, $n \in \mathbb{N}$, define a sequence of elements of $\mathbb{Q}$. We have $|\alpha_n - a|_p \leqslant p^{-(n+1)}$, that is, $\alpha_n \to a$. Moreover, the sequence (α_n) of rational numbers is Cauchy with respect to $|\ |_p$, since for every $M \in \mathbb{N}$ we have $|\alpha_m - \alpha_n|_p \leqslant p^{-(M+1)}$ for all $m, n \geqslant M$. Thus $\varphi : \mathbb{Q}_p \to L$, $a \mapsto (\alpha_n) + \mathfrak{m}$, is a well-defined field homomorphism. Being a field homomorphism φ is injective.

What remains is to show that the map φ is surjective. Take any $(\beta_n) + \mathfrak{m} \in L$. Since (β_n) is a Cauchy sequence, by Theorem 2.61 it converges to a point $a \in \mathbb{Q}_p$. We construct the sequence (α_n) corresponding to a as described in the last paragraph. Then $\alpha_n \to a$ as well and hence using the triangle inequality (or the non-Archimedean condition) we have $\alpha_n - \beta_n = (\alpha_n - a) - (\beta_n - a) \to 0$, that is, $(\alpha_n) - (\beta_n) = (\alpha_n - \beta_n) \in \mathfrak{m}$, that is, $\varphi(a) = (\alpha_n) + \mathfrak{m} = (\beta_n) + \mathfrak{m}$. ■

Corollary 2.29 The p-adic series $\sum_{n\in\mathbb{N}} a_n$ (with $a_n \in \mathbb{Q}_p$) converges if and only if $|a_n|_p \to 0$.

Proof The *only if* part is obvious. For the *if* part, take a sequence (a_n) of p-adic numbers with $|a_n|_p \to 0$. Define $s_n := \sum_{i=1}^{n} a_i$. Since $a_{n+1} = s_{n+1} - s_n \to 0$ by hypothesis, Lemma 2.18 guarantees that (s_n) is a Cauchy sequence, that is, (s_n) converges in $\mathbb{Q}_p$. ■

This is quite unlike the Archimedean norm $|\ |_\infty$. For example, with respect to this norm $\frac{1}{n} \to 0$, whereas the series $\sum_{n\in\mathbb{N}} \frac{1}{n}$ diverges.

2.14.3 Hensel's Lemma

Let us conclude our short study of p-adic methods by proving an important theorem due to Hensel. This theorem talks about the solvability of polynomial equations $f(X) = 0$ for $f(X) \in \hat{\mathbb{Z}}_p[X]$. Before proceeding further, let us introduce a notation. Recall that every $x \in \hat{\mathbb{Z}}_p$ has a unique p-adic expansion of the form $a = a_0 + a_1 p + a_2 p^2 + \cdots$ with $0 \leqslant a_n < p$ (Exercises 2.144 and 2.145). If $a_0 = a_1 = \cdots = a_{n-1} = 0$, then $a = a_n p^n + a_{n+1} p^{n+1} + a_{n+2} p^{n+2} + \cdots = p^n b$, where $b := a_n + a_{n+1} p + a_{n+2} p^2 + \cdots \in \hat{\mathbb{Z}}_p$. Thus $p^n | a$ in $\hat{\mathbb{Z}}_p$. We denote this by saying that $a \equiv 0 \pmod{p^n}$. Notice that $a \equiv 0 \pmod{p^n}$ if and only if $|a|_p \leqslant p^{-n}$. We write $a \equiv b \pmod{p^n}$ for $a, b \in \hat{\mathbb{Z}}_p$, if $a - b \equiv 0 \pmod{p^n}$. Since p^n can be viewed as the element $\iota(p^n)$ of $\hat{\mathbb{Z}}_p$, this congruence notation conforms to that for a general PID. ($\hat{\mathbb{Z}}_p$ is a PID by Exercise 2.148.)

Since by our assumption any ring A comes with identity (that we denote by $1 = 1_A$), it makes sense to talk for every $n \in \mathbb{Z}$ about an element $n = n_A$ in A, which is the n-fold sum of 1. More precisely:

$$n := \begin{cases} 0 & \text{if } n = 0, \\ 1 + 1 + \cdots + 1 \ (n \text{ times}) & \text{if } n > 0, \\ -(-n) & \text{if } n < 0. \end{cases}$$

Given any $f(X) = a_0 + a_1 X + a_2 X^2 + \cdots + a_d X^d \in A[X]$, one can define the *formal derivative* of f as $f'(X) := a_1 + 2a_2 X + \cdots + d a_d X^{d-1} \in A[X]$. Properties of formal derivatives of polynomials are covered in Exercise 2.61.

Theorem 2.63 **Hensel's lemma** Let $f(X) \in \hat{\mathbb{Z}}_p[X]$. Suppose that there exist $M \in \mathbb{N}_0$ and $\alpha_0 \in \hat{\mathbb{Z}}_p$ satisfying:

(1) $|f(\alpha_0)|_p \leqslant p^{-(2M+1)}$ (that is, α_0 is a solution of $f(x) \equiv 0 \pmod{p^{2M+1}}$), and
(2) $|f'(\alpha_0)|_p = p^{-M}$ (that is, $f'(\alpha_0) \not\equiv 0 \pmod{p^{M+1}}$).

Then there exists a unique $\alpha \in \hat{\mathbb{Z}}_p$ such that $f(\alpha) = 0$ and $|\alpha - \alpha_0|_p \leqslant p^{-(M+1)}$ (that is, $\alpha \equiv \alpha_0 \pmod{p^{M+1}}$).

Proof Let us inductively construct a sequence $\alpha_0, \alpha_1, \alpha_2, \ldots$ of p-adic integers with the properties that $|f(\alpha_n)|_p \leqslant p^{-(2M+n+1)}$ and $|f'(\alpha_n)|_p = p^{-M}$ for every $n \in \mathbb{N}_0$. The given α_0 provides the starting point (induction basis). For the inductive step, assume that $n \geqslant 1$ and that $\alpha_0, \alpha_1, \ldots, \alpha_{n-1}$ have been constructed with the desired properties. we now explain how to construct α_n from α_{n-1}. Put

$$\alpha_n := \alpha_{n-1} + k_n p^{M+n} \quad \text{for some } k_n \in \hat{\mathbb{Z}}_p\,.$$

We want to find a suitable k_n for which $|f(\alpha_n)|_p \leqslant p^{-(2M+n+1)}$. Taylor expansion gives $f(\alpha_n) = f(\alpha_{n-1}) + k_n p^{M+n} f'(\alpha_{n-1}) + c_n p^{2(M+n)}$ for some $c_n \in \hat{\mathbb{Z}}_p$. Since by induction hypothesis $p^{2M+n} | f(\alpha_{n-1})$ and $p^M | f'(\alpha_{n-1})$, we can write

$$f(\alpha_n) = p^{2M+n} \left(\frac{f(\alpha_{n-1})}{p^{2M+n}} + k_n \frac{f'(\alpha_{n-1})}{p^M} + c_n p^n \right).$$

Since $p^{M+1} \not| f'(\alpha_{n-1})$, the element $\frac{f'(\alpha_{n-1})}{p^M} \in U_p$ and, therefore, there is a unique solution for k_n of the congruence

$$\frac{f(\alpha_{n-1})}{p^{2M+n}} + k_n \frac{f'(\alpha_{n-1})}{p^M} \equiv 0 \pmod{p}.$$

This value of k_n yields

$$f(\alpha_n) = p^{2M+n}(b_n p + c_n p^n) \equiv 0 \pmod{p^{2M+n+1}}$$

for some $b_n \in \hat{\mathbb{Z}}_p$. The Taylor expansion of f' gives $f'(\alpha_n) = f'(\alpha_{n-1}) + d_n p^{M+n}$ (for some $d_n \in \hat{\mathbb{Z}}_p$) which implies that $f'(\alpha_n) \equiv f'(\alpha_{n-1}) \pmod{p^M}$, that is, $|f'(\alpha_n)|_p = p^{-M}$.

Since $|\alpha_n - \alpha_{n-1}|_p \leqslant p^{-(M+n)}$, it follows that $\alpha_n - \alpha_{n-1} \to 0$, that is, (α_n) is a Cauchy sequence (under $|\ |_p$). By the completeness of $\mathbb{Q}_p$, we then have an $\alpha \in \mathbb{Q}_p$ such that $\alpha_n \to \alpha$. Similarly $f(\alpha_n) - f(\alpha_{n-1}) \to 0$, that is, the sequence $(f(\alpha_n))$ is Cauchy and hence converges to $f(\alpha)$. Also $|f(\alpha_n)|_p \leqslant p^{-(2M+n+1)}$, that is, $f(\alpha_n) \to 0$, that is, $f(\alpha) = 0$. Finally, each $\alpha_n \equiv \alpha_0 \pmod{p^{M+1}}$, so that $\alpha \equiv \alpha_0 \pmod{p^{M+1}}$. This establishes the existence of a desired $\alpha \in \mathbb{Q}_p$.

For proving the uniqueness of α, let $\beta \in \mathbb{Q}_p$ satisfy $f(\beta) = 0$ and $|\beta - \alpha_0|_p \leqslant p^{-(M+1)}$. By Taylor expansion, $f(\beta) = f(\alpha) + (\beta - \alpha) f'(\alpha) + (\beta - \alpha)^2 c$ for some $c \in \hat{\mathbb{Z}}_p$, that is, $(\beta - \alpha)(f'(\alpha) + (\beta - \alpha)c) = 0$. Now $\beta - \alpha = (\beta - \alpha_0) - (\alpha - \alpha_0)$ and so $|\beta - \alpha|_p \leqslant \max(|\beta - \alpha_0|_p, |\alpha - \alpha_0|_p) \leqslant p^{-(M+1)}$, whereas $f'(\alpha_n) \to f'(\alpha)$, so that $|f'(\alpha)|_p = p^{-M}$. Therefore, $f'(\alpha) + (\beta - \alpha)c \not\equiv 0 \pmod{p^{M+1}}$ and, in particular, $f'(\alpha) + (\beta - \alpha)c \neq 0$. Thus we must have $\beta - \alpha = 0$. ■

Note that α_n in the last proof satisfies the congruence

$$f(\alpha_n) \equiv 0 \pmod{p^{2M+n+1}}$$

for each $n \in \mathbb{N}_0$. We are given the solution α_0 corresponding to $n = 0$. From this, we inductively construct the solutions $\alpha_1, \alpha_2, \ldots$ corresponding to $n = 1, 2, \ldots$ respectively. The process for computing α_n from α_{n-1} as described in the proof of Hensel's lemma is referred to as *Hensel lifting*. The given conditions ensure that this lifting is possible (and uniquely doable) for every $n \in \mathbb{N}$, and in the limit $n \to \infty$ we get a root $\alpha \in \hat{\mathbb{Z}}_p$ of f. Since each k_n is required modulo p, we can take $k_n \in \{0, 1, \ldots, p-1\}$. So α admits a p-adic expansion of the form $\alpha = \alpha_0 + k_1 p^{M+1} + k_2 p^{M+2} + k_3 p^{M+3} + \cdots$.

The special case $M = 0$ for Hensel's lemma is now singled out:

Corollary 2.30 Let $f(X) \in \hat{\mathbb{Z}}_p[X]$. Suppose that there exists an $\alpha_0 \in \hat{\mathbb{Z}}_p$ satisfying:

(1) $|f(\alpha_0)|_p < 1$ (that is, α_0 is a solution of $f(x) \equiv 0 \pmod{p}$), and

(2) $|f'(\alpha_0)|_p = 1$ (that is, $f'(\alpha_0) \not\equiv 0 \pmod{p}$, that is, α_0 is a *simple root* of f modulo p).

Then there exists a unique $\alpha \in \hat{\mathbb{Z}}_p$ such that $f(\alpha) = 0$ and $|\alpha - \alpha_0|_p < 1$ (that is, $\alpha \equiv \alpha_0 \pmod{p}$). ■

For this special case, we compute solutions α_n of $f(x) \equiv 0 \pmod{p^{n+1}}$ inductively for $n = 1, 2, 3, \ldots$, given a suitable solution α_0 of this congruence for $n = 0$. The lifting formula is now:

$$\alpha_n = \alpha_{n-1} + k_n p^n, \text{ where } k_n \equiv -f'(\alpha_{n-1})^{-1} \left(\frac{f(\alpha_{n-1})}{p^n} \right) \pmod{p}. \qquad \textbf{(2.21)}$$

Example 2.35 $\mathbb{Z}$ is canonically embedded in $\hat{\mathbb{Z}}_p$ and so is $\mathbb{Z}[X]$ in $\hat{\mathbb{Z}}_p[X]$. Thus it makes sense to carry out the lifting process for a polynomial $f(X) \in \mathbb{Z}[X]$ and for some solution α_0 of $f(X) \equiv 0 \pmod{p}$ in $\mathbb{Z}$. One solves Formula (2.21) in $\mathbb{Z}$ and obtains each $\alpha_n \in \mathbb{Z}$. The limit α belongs to $\hat{\mathbb{Z}}_p$ and is a solution of $f(X) = 0$ in $\hat{\mathbb{Z}}_p$.

For example, let p be an *odd* prime and $\left(\frac{a}{p}\right) = 1$. Let $\alpha_0 \in \mathbb{Z}$ be a solution of $X^2 \equiv a \pmod{p}$. Here $f(X) = X^2 - a$, so that $f'(X) = 2X$, that is, $f'(\alpha_0) = 2\alpha_0 \not\equiv 0 \pmod{p}$. Thus the conditions of Corollary 2.30 are satisfied and we get a unique square root of α in $\hat{\mathbb{Z}}_p$ with $\alpha \equiv \alpha_0 \pmod{p}$. This α has a p-adic expansion of the form $\alpha = \alpha_0 + k_1 p + k_2 p^2 + k_3 p^3 + \cdots$.

As a specific numerical example, take $p = 7$, $a = 2$ and $\alpha_0 = 3$. Using Formula (2.21), we compute $k_1 = 1$, $\alpha_1 = 10$, $k_2 = 2$, $\alpha_2 = 108$, $k_3 = 6$, $\alpha_3 = 2166$, and so on. Thus a square root of 2 in $\hat{\mathbb{Z}}_7$ is $3 + 1 \times 7 + 2 \times 7^2 + 6 \times 7^3 + \cdots$. The other square root of α in $\hat{\mathbb{Z}}_7$ can be obtained by starting with $\alpha_0 = 4$. □

EXERCISE SET 2.14

2.144 (a) Establish that any p-adic integer (a_n) can be *uniquely* described as a sequence $(x_n)_{n \in \mathbb{N}_0}$ of integers x_n satisfying $0 \leqslant x_n < p$ for every $n \in \mathbb{N}_0$ and $a_n \equiv x_0 + x_1 p + \cdots + x_{n-1} p^{n-1} \pmod{p^n}$ for every $n \in \mathbb{N}$. In this case, the p-adic integer (a_n) is written as the *infinite series*

$$(a_n) = x_0 + x_1 p + x_2 p^2 + \cdots .$$

One calls the above series the *p-adic expansion* of (a_n). Note that the sum in the above series is not to be treated as one of integers. However, for $a \in \mathbb{N}_0$ the expansion of a to the base p is the same as the p-adic expansion of a (more correctly of $\iota(a) = (a)$). In other words, if the p-adic expansion of (a_n) is terminating, that is, $x_N = x_{N+1} = x_{N+2} = \cdots = 0$ for some N, then (a_n) can be identified with the rational integer $x_0 + x_1 p + \cdots + x_{N-1} p^{N-1}$. A non-terminating p-adic series, on the other hand, diverges under the Archimedean norm, but converges under the p-adic norm and corresponds to an element of $\hat{\mathbb{Z}}_p$ not in $\mathbb{N}_0$. The rational integer -1, for example, has the infinite p-adic expansion $(p-1) + (p-1)p + (p-1)p^2 + \cdots$. The sum telescopes and in the limit $n \to \infty$ converges (under the p-adic norm) to $\lim_{n \to \infty} p^n - 1 = -1$.

(b) Let $a \in \mathbb{N}$. Write the p-adic expansion for $-a$. **[H]**
(c) Given p-adic integers $a := x_0 + x_1p + x_2p^2 + \cdots$ and $b := y_0 + y_1p + y_2p^2 + \cdots$, find the p-adic integers $c := z_0 + z_1p + z_2p^2 + \cdots$ and $d := w_0 + w_1p + w_2p^2 + \cdots$, such that $c = a + b$ and $d = ab$. (Express each z_n and w_n explicitly in terms of x_n's and y_n's.)

2.145 In view of Exercise 2.144, every $x \in \hat{\mathbb{Z}}_p$ admits a unique expansion of the form $x = x_0 + x_1p + x_2p^2 + \cdots$, where each $x_i \in \{0, 1, \ldots, p-1\}$. This notion of *$p$-adic expansion* can be extended to the elements of $\mathbb{Q}_p$.
(a) Show that for $x \in \mathbb{Q}_p \setminus \hat{\mathbb{Z}}_p$, there exist unique $r \in \mathbb{N}$ and unique integers x_{-r}, $x_{-r+1}, \ldots, x_{-1}, x_0, x_1, \ldots$, each in $\{0, 1, \ldots, p-1\}$, such that x can be written as:

$$x = x_{-r}p^{-r} + x_{-r+1}p^{-r+1} + \cdots + x_{-1}p^{-1} + x_0 + x_1p + x_2p^2 + \cdots .$$

(b) Describe how to compute the p-adic expansions of $x + y$ and xy given those for $x, y \in \mathbb{Q}_p$. Also of x/y provided that $y \neq 0$.
(c) What is $|x|_p$ for $x := x_0 + x_1p + x_2p^2 + \cdots \in \hat{\mathbb{Z}}_p$?
(d) What is $|x|_p$ for $x := x_{-r}p^{-r} + x_{-r+1}p^{-r+1} + \cdots + x_{-1}p^{-1} + x_0 + x_1p + x_2p^2 + \cdots \in \mathbb{Q}_p$ with $x_{-r} \neq 0$.

2.146 Let p be an odd prime and $a \in \mathbb{Z}$ with $\left(\frac{a}{p}\right) = 1$. From elementary number theory we know that the congruence $x^2 \equiv a \pmod{p^n}$ has two solutions for every $n \in \mathbb{N}$. Let x_1 be a solution of $x^2 \equiv a \pmod{p}$. We know that a solution x_n of $x^2 \equiv a \pmod{p^n}$ *lifts* uniquely to a solution x_{n+1} of $x^2 \equiv a \pmod{p^{n+1}}$. Thus we can inductively compute a sequence $x_1, x_2, x_3, \ldots$ of integers. Show that (x_n) is a p-adic integer and that $(x_n)^2 = (a)$.

2.147 **(a)** Show that the ring $\hat{\mathbb{Z}}_p$ *contains* rationals of the form a/b, $a, b \in \mathbb{Z}$, $p \nmid b$. This implies that $\mathbb{Z} \subsetneqq \hat{\mathbb{Z}}_p$.
(b) Take $a := 17$ for $p = 2$, $a := 7$ for $p = 3$ and $a := p + 1$ for $p > 3$. Show that there exists $x \in \mathbb{Q}_p$ with $x^2 = a$ in $\mathbb{Q}_p$. Show also that such an x does not belong to $\mathbb{Q}$. Thus $\mathbb{Q} \subsetneqq \mathbb{Q}_p$.
(c) Show that $1/p \in \mathbb{Q}_p \setminus \hat{\mathbb{Z}}_p$. Thus $\hat{\mathbb{Z}}_p \subsetneqq \mathbb{Q}_p$.

2.148 Prove the following assertions:
(a) $\hat{\mathbb{Z}}_p = \{x \in \mathbb{Q}_p \mid |x|_p \leqslant 1\}$.
(b) $U_p = \{x \in \mathbb{Q}_p \mid |x|_p = 1\}$.
(c) Every non-zero ideal of $\hat{\mathbb{Z}}_p$ is of the form $\mathfrak{a}_r := \{x \in \hat{\mathbb{Z}}_p \mid |x|_p \leqslant p^{-r}\}$ for some $r \in \mathbb{N}_0$.
(d) The ideals $\mathfrak{a}_r$ of Part (c) satisfy the infinite strictly descending chain $\hat{\mathbb{Z}}_p = \mathfrak{a}_0 \supsetneqq \mathfrak{a}_1 \supsetneqq \mathfrak{a}_2 \supsetneqq \cdots$.
(e) $\hat{\mathbb{Z}}_p$ is a local domain with the maximal ideal $\mathfrak{m}_p := \mathfrak{a}_1 = \{x \in \hat{\mathbb{Z}}_p \mid |x|_p < 1\}$.
(f) The ideal $\mathfrak{a}_r$ of Part (c) is the principal ideal of $\hat{\mathbb{Z}}_p$ generated by p^r, and $\hat{\mathbb{Z}}_p/\mathfrak{a}_r \cong \mathbb{Z}/p^r\mathbb{Z}$. In particular, $\hat{\mathbb{Z}}_p$ is a local PID, that is, a discrete valuation domain (Exercise 2.133), with the residue field $\hat{\mathbb{Z}}_p/\mathfrak{m}_p \cong \mathbb{F}_p$.

2.149 Compute the p-adic expansion of $1/3$ in $\mathbb{Q}_5$ and of $-2/5$ in $\mathbb{Q}_3$.

2.150 Show that $\mathbb{Z}$ is dense in $\hat{\mathbb{Z}}_p$ under the p-adic norm $|\ |_p$, that is, show that given any $x \in \hat{\mathbb{Z}}_p$ and real $\epsilon > 0$, there exists $a \in \mathbb{Z}$ with $|x - a|_p < \epsilon$. Show also that $\mathbb{Q}$ is dense in $\mathbb{Q}_p$.

2.151 Prove the following assertions that establish that $\hat{\mathbb{Z}}_p$ is the *closure* of $\mathbb{Z}$ in $\mathbb{Q}_p$ under $|\ |_p$.

(a) Every sequence (a_n) of rational integers, Cauchy under $|\ |_p$, converges in $\hat{\mathbb{Z}}_p$.

(b) If a sequence (a_n) of rational numbers, Cauchy under $|\ |_p$, converges to $x \in \hat{\mathbb{Z}}_p$, then there exists a sequence (b_n) of rational *integers*, Cauchy under $|\ |_p$, that converges to x.

2.152 Show that:

(a) The series $\sum_{n\in\mathbb{N}} n!$ converges in $\mathbb{Q}_p$.

(b) The series $\sum_{n\in\mathbb{N}} n \cdot n!$ converges in $\mathbb{Q}_p$.

(c) $\sum_{n\in\mathbb{N}} n \cdot n! = -1$ in $\mathbb{Q}_p$. **[H]**

(d) The series $\sum_{n\in\mathbb{N}} \frac{1}{n}$ does not converge in $\mathbb{Q}_p$.

(e) If $a \in \mathbb{Q}_p$ and $|a|_p < 1$, then $\frac{1}{1-a} = 1 + a + a^2 + \cdots$.

2.153 Prove that $\displaystyle\prod_{p\,\in\,\mathbb{P}\,\cup\,\{\infty\}} |a|_p = 1$ for any non-zero $a \in \mathbb{Q}$. **[H]**

2.154 Prove that for any $a \in \hat{\mathbb{Z}}_p$ the sequence (a^{p^n}) converges in $\mathbb{Q}_p$. **[H]**

2.155 Let $p, q \in \mathbb{P}$, $p \neq q$. Show that the fields $\mathbb{Q}_p$ and $\mathbb{Q}_q$ are not isomorphic.

2.156 Let a be an integer congruent to 1 modulo 8. Show that there exists an $\alpha \in \hat{\mathbb{Z}}_2$ such that $\alpha^2 = a$ and $|\alpha - 1|_2 \leqslant \frac{1}{4}$.

2.157 Compute $\alpha \in \hat{\mathbb{Z}}_3$ with $\alpha^2 + \alpha + 223 = 0$ and $\alpha \equiv 4 \pmod{243}$.

2.158 Let p be an odd prime and $a \in \{0, 1, \ldots, p-1\}$. Show that the polynomial $X^2 - a$ has exactly $1 + \left(\frac{a}{p}\right)$ roots in $\hat{\mathbb{Z}}_p$.

2.159 Show that the polynomial $X^2 - p$ is irreducible in $\hat{\mathbb{Z}}_p[X]$.

2.160 **Teichmüller representative** Let $a \in \{0, 1, \ldots, p-1\}$. Show that there exists a unique $\alpha \in \hat{\mathbb{Z}}_p$ such that $\alpha^p = \alpha$ and $\alpha \equiv a \pmod{p}$.

2.161 Show that the algebraic closure $\bar{\mathbb{Q}}_p$ of $\mathbb{Q}_p$ is of infinite extension degree over $\mathbb{Q}_p$. **[H]**

2.15 Statistical Methods

Many attacks on cryptosystems involve statistical analysis of ciphertexts and also of data collected from the victim's machine during one or more private-key operations. For a proper understanding of these analysis techniques, one requires some knowledge of statistics and random variables. In this section, we provide a quick overview of some statistical gadgets. We make the assumption that the reader is already familiar with the elementary notion of probability. We denote the probability of an event E by $\Pr(E)$.

2.15.1 Random Variables and Their Probability Distributions

An experiment whose outcome is random is referred to as a *random experiment*. The set of all possible outcomes of a random experiment is called the *sample space* of the experiment. For example, the outcomes of tossing a coin can be mapped to the set $\{H, T\}$ with H and T standing respectively for *head* and *tail*. It is convenient to assign *numerical* values to the outcomes of a random experiment. Identifying *head* with 0 and *tail* with 1, one can view coin tossing as a random experiment with sample space $\{0, 1\}$. Some other random experiments include throwing a die (with sample space $\{1, 2, 3, 4, 5, 6\}$), the life of an electric bulb (with sample space $\mathbb{R}_+$, the set of all non-negative real numbers), and so on. Unless otherwise specified, we henceforth assume that sample spaces are subsets of $\mathbb{R}$.

A *random variable* is a variable which can assume (all and only) the values from a (given) sample space.

A *discrete random variable* can assume only countably many values, that is, the sample space S_X of a discrete random variable X either is finite or has a bijection with $\mathbb{N}$, that is, we can *enumerate* the elements of S_X as $x_1, x_2, x_3, \ldots$.

The *probability distribution function* or the *probability mass function*

$$f_X : S_X \to [0, 1]$$

of a discrete random variable X assigns to each x in the sample space S_X of X the probability of the occurrence of the value x in a random experiment.[21] We have

$$\sum_{x \in S_X} f_X(x) = 1 \,.$$

A *continuous random variable* assumes uncountable number of values, that is, the sample space S_X of a continuous random variable X cannot be in bijective correspondence with a subset of $\mathbb{N}$. Typically S_X is an interval $[a, b]$ or (a, b) with $-\infty \leqslant a < b \leqslant +\infty$.

One does not assign individual probabilities $\Pr(X = x)$ to a value assumed by a continuous random variable X.[22] The probabilistic behaviour of X is in this case described by the *probability density function*

$$f_X : S_X \to \mathbb{R}_+$$

[21] $[a, b]$ is the *closed interval* consisting of all real numbers u satisfying $a \leqslant u \leqslant b$. Similarly, the *open interval* (a, b) is the set of all real values u satisfying $a < u < b$. In order to make a distinction between the open interval (a, b) and the ordered pair (a, b), many—mostly Europeans—use the notation $]a, b[$ for denoting open intervals.

[22] More correctly, $\Pr(X = x) = 0$ for each $x \in S_X$.

with the implication that the probability that X occurs in the interval $[c, d]$ (or (c, d)) is given by the integral

$$\Pr(c \leqslant X \leqslant d) = \int_c^d f_X(x)\,\mathrm{d}x\,,$$

that is, by the area between the x-axis, the curve $f_X(x)$ and the vertical lines $x = c$ and $x = d$. We have

$$\int_{x \in S_X} f_X(x)\,\mathrm{d}x = 1\,.$$

It is sometimes useful to *set* $f_X(x) := 0$ for $x \in \mathbb{R} \setminus S_X$, so that f_X is defined on the entire real line $\mathbb{R}$.

The *cumulative probability distribution* $F_X : \mathbb{R} \to [0, 1]$ of a random variable X (discrete or continuous) is the function $F_X(x) := \Pr(X \leqslant x)$ for all $x \in \mathbb{R}$. If X is continuous, we have

$$F_X(x) = \int_{-\infty}^x f_X(x)\,\mathrm{d}x\,,$$

which implies that

$$f_X(x) = \frac{\mathrm{d}}{\mathrm{d}x} F_X(x)\,.$$

2.15.2 Operations on Random Variables

Let X and Y be discrete random variables. The *joint probability distribution* of X, Y refers to a random variable Z with $S_Z = S_X \times S_Y$. For $z = (x, y)$, the probability of $Z = z$ is denoted by $f_Z(z) = \Pr(Z = z) = \Pr(X = x, Y = y)$. The probability $\Pr(X = x, Y = y)$ stands for the probability that $X = x$ *and* $Y = y$. The random variables X and Y are called *independent*, if

$$\Pr(X = x, Y = y) = \Pr(X = x)\Pr(Y = y)$$

for all x, y.

Example 2.36 Suppose that we have an urn containing three identical balls with labels $1, 2, 3$. We draw two balls randomly from the urn. Let us denote the outcome of the first drawing by X and that of the second drawing by Y. We consider the joint distribution X, Y of the two outcomes in the two following cases:

(1) The balls are drawn *with replacement*, that is, after the first ball is drawn, it is returned back to the urn (and the urn is shaken well), before the next ball is drawn. The joint probability distribution is now as follows:

$$\Pr(X = x, Y = y) = \frac{1}{9} \quad \text{for all } x, y \in \{1, 2, 3\}\,.$$

In this case, the outcome of the second drawing is not influenced by the outcome of the first drawing; that is, X and Y are independent, and we have $\Pr(X = x, Y = y) = \frac{1}{9} = \frac{1}{3} \times \frac{1}{3} = \Pr(X = x)\Pr(Y = y)$, as expected.

(2) The balls are drawn *without replacement*, that is, the ball obtained by the first drawing is *not* returned to the urn, before the second ball is drawn. In this case, the outcome of the second drawing is influenced by that of the first drawing in the sense that the same ball cannot be drawn on both occasions. Thus, X and Y are now dependent. This is revealed by the following joint probability distribution:

x	y	$\Pr(X = x, Y = y)$
1	1	0
1	2	1/6
1	3	1/6
2	1	1/6
2	2	0
2	3	1/6
3	1	1/6
3	2	1/6
3	3	0

□

For continuous random variables X and Y, the joint distribution is defined by the probability density function $f_{X,Y}(x, y)$ and the cumulative distribution is obtained by the double integral

$$F_{X,Y}(c, d) = \int_{x=-\infty}^{c} \int_{y=-\infty}^{d} f_{X,Y}(x, y)\, \mathrm{d}x\, \mathrm{d}y\,.$$

X and Y are independent, if $f_{X,Y}(x, y) = f_X(x) f_Y(y)$ for all x, y. In this case, we also have $F_{X,Y}(c, d) = F_X(c) F_Y(d)$ for all c, d.

Now, we define arithmetic operations on random variables. First, let X and Y be discrete random variables. The sum $X + Y$ is defined to be a random variable U which assumes the values $u = x + y$ for $x \in S_X$ and $y \in S_Y$ with probability

$$f_U(u) = \Pr(U = u) = \sum_{\substack{x,y \\ x+y=u}} \Pr(X = x, Y = y)\,.$$

The product XY of X and Y is defined to be a random variable V which assumes the values $v = xy$ for $x \in S_X$ and $y \in S_Y$ with probability

$$f_V(v) = \Pr(V = v) = \sum_{\substack{x,y \\ xy=v}} \Pr(X = x, Y = y)\,.$$

For $\alpha \in \mathbb{R}^*$, the random variable $W = \alpha X$ assumes the values $w = \alpha x$ for $x \in S_X$ with probability

$$f_W(w) = \Pr(W = \alpha x) = \Pr(X = x) = f_X(x)\,.$$

Example 2.37 Let us consider the random variables X and Y of Example 2.36. For the sake of brevity, we denote $\Pr(X = x, Y = y)$ by P_{xy}. The distributions of $U = X + Y$ in the two cases are as follows:

(1) Drawing with replacement:

$$\begin{aligned}
\Pr(U=2) &= P_{11} = 1/9 \\
\Pr(U=3) &= P_{12}+P_{21} = 2/9 \\
\Pr(U=4) &= P_{13}+P_{22}+P_{31} = 1/3 \\
\Pr(U=5) &= P_{23}+P_{32} = 2/9 \\
\Pr(U=6) &= P_{33} = 1/9
\end{aligned}$$

(2) Drawing without replacement:

$$\begin{aligned}
\Pr(U=3) &= P_{12}+P_{21} = 1/3 \\
\Pr(U=4) &= P_{13}+P_{31} = 1/3 \\
\Pr(U=5) &= P_{23}+P_{32} = 1/3
\end{aligned}$$

□

Now, let us consider continuous random variables X and Y. In this case, it is easier to define first the cumulative density functions of $U = X+Y$, $V = XY$ and $W = \alpha X$ and then the probability density functions by taking derivatives:

$$\begin{aligned}
F_U(u) &:= \int\int_{x+y\leqslant u} f_{X,Y}(x,y)\,\mathrm{d}x\,\mathrm{d}y = \int_{x=-\infty}^{\infty}\int_{y=-\infty}^{u-x} f_{X,Y}(x,y)\,\mathrm{d}x\,\mathrm{d}y \\
F_V(v) &:= \int\int_{xy\leqslant v} f_{X,Y}(x,y)\,\mathrm{d}x\,\mathrm{d}y \\
F_W(w) &:= \begin{cases} \displaystyle\int_{-\infty}^{w/\alpha} f_X(x)\,\mathrm{d}x & \text{if } \alpha > 0 \\ \displaystyle\int_{w/\alpha}^{\infty} f_X(x)\,\mathrm{d}x & \text{if } \alpha < 0 \end{cases}
\end{aligned}$$

One can easily generalize sums and products to an arbitrary *finite* number of random variables. More generally, if $X_1,\ldots,X_n$ are random variables and $g(X_1,\ldots,X_n) \in \mathbb{R}[X_1,\ldots,X_n]$, one can talk about the probability distribution or density function of the random variable $g(X_1,\ldots,X_n)$. (See Exercise 2.163.)

Now, we introduce the important concept of *conditional probability*. Let X and Y be two random variables. To start with, suppose that they are discrete. We denote by $f(x,y) = \Pr(X=x, Y=y)$ the joint probability distribution function of X, Y. For $y \in S_Y$ with $\Pr(Y=y) > 0$, we define the conditional probability of $X = x$ *given* $Y = y$ as:

$$f_{X|y}(x) := \Pr(X=x|Y=y) := \frac{\Pr(X=x, Y=y)}{\Pr(Y=y)} = \frac{f(x,y)}{f_Y(y)}\,.$$

For a fixed $y \in S_Y$, the probabilities $f_{X|y}(x)$, $x \in S_X$, constitute the probability distribution function of the random variable $X|y$ (X given $Y = y$). If X and Y are independent, $f(x,y) = f_X(x)f_Y(y)$ and so $f_{X|y}(x) = f_X(x)$ for all $x \in S_X$, that is, the random variables X and $X|y$ have the same probability distribution. This is

expected, because in this case the probability of $X = x$ does not depend on whatever value y the variable Y takes.

If X and Y are continuous random variables with joint density $f(x, y)$ and $y \in S_Y$, the conditional probability density function of $X|y$ (X given $Y = y$) is defined by

$$f_{X|y}(x) := \frac{f(x,y)}{f_Y(y)}\,.$$

Again if X and Y are independent, we have $f_{X|y}(x) = f_X(x)$ for all x, y.

For a fixed $x \in S_X$, one can likewise define the conditional probabilities $f_{Y|x}(y) := f(x, y)/f_X(x)$ for all $y \in S_Y$.

Let X and Y be discrete random variables with joint distribution $f(x, y)$. Also let $\Gamma \subseteq S_X$ and $\Delta \subseteq S_Y$. One defines the probability $f_X(\Gamma)$ as:

$$f_X(\Gamma) := \Pr(\Gamma) := \Pr(x \in \Gamma) = \sum_{x\in\Gamma} f_X(x).$$

The joint probability $f(\Gamma, \Delta)$ is defined as:

$$f(\Gamma,\Delta) := \Pr(\Gamma,\Delta) := \sum_{x\in\Gamma,\, y\in\Delta} \Pr(X=x, Y=y) \;=\; \sum_{x\in\Gamma,\, y\in\Delta} f(x,y)\,.$$

If $\Gamma = \{x\}$ is a singleton, we prefer to write $f(x, \Delta)$ instead of $f(\{x\}, \Delta)$. Similarly, $f(\Gamma, y)$ stands for $f(\Gamma, \{y\})$. We also define the conditional distributions:

$$f_{X|\Delta}(x) \;:=\; \Pr(x|\Delta) \;:=\; \tfrac{f(x,\Delta)}{f_Y(\Delta)}\,, \quad \text{provided that } f_Y(\Delta) > 0,$$

$$f_{Y|\Gamma}(y) \;:=\; \Pr(y|\Gamma) \;:=\; \tfrac{f(\Gamma,y)}{f_X(\Gamma)}\,, \quad \text{provided that } f_X(\Gamma) > 0.$$

We abbreviate $f_{X|\Delta}(\Gamma)$ as $\Pr(\Gamma|\Delta)$ and $f_{Y|\Gamma}(\Delta)$ as $\Pr(\Delta|\Gamma)$.

Theorem 2.64 **Bayes rule** Let X, Y be discrete random variables and $\Delta \subseteq S_Y$ with $f_Y(\Delta) > 0$. Also let $\Gamma_1, \ldots, \Gamma_n$ form a partition of S_X with $f_X(\Gamma_i) > 0$ for all $i = 1, \ldots, n$. Then we have:

$$f_{X|\Delta}(\Gamma_i) = \frac{f_{Y|\Gamma_i}(\Delta) f_X(\Gamma_i)}{\sum_{j=1}^n f_{Y|\Gamma_j}(\Delta) f_X(\Gamma_j)}\,,$$

that is, in terms of probability:

$$\Pr(\Gamma_i|\Delta) = \frac{\Pr(\Delta|\Gamma_i)\Pr(\Gamma_i)}{\sum_{j=1}^n \Pr(\Delta|\Gamma_j)\Pr(\Gamma_j)}\,.$$

Proof $\Pr(\Gamma_i, \Delta) = \Pr(\Delta|\Gamma_i)\Pr(\Gamma_i) = \Pr(\Gamma_i|\Delta)\Pr(\Delta)$. So it is sufficient to show that $\Pr(\Delta)$ equals the sum in the denominator. The event Δ is the union of the pairwise disjoint events (Γ_j, Δ), $j = 1, \ldots, n$, and so $\Pr(\Delta) = \sum_{j=1}^n \Pr(\Gamma_j, \Delta) = \sum_{j=1}^n \Pr(\Delta|\Gamma_j)\Pr(\Gamma_j)$. ■

The Bayes rule relates the a priori probabilities $\Pr(\Gamma_j)$ and $\Pr(\Delta|\Gamma_j)$ to the a posteriori probabilities $\Pr(\Gamma_i|\Delta)$. The following example demonstrates this terminology.

Example 2.38 Consider the random experiment of Example 2.36(2). Take $\Gamma_j := \{j\}$ for $j \in \{1,2,3\}$ and $\Delta := \{2,3\}$. We have the following a priori probabilities:

$$\begin{aligned}
\Pr(\Gamma_j) &= \text{Probability of getting ball } j \text{ in the first draw } = 1/3,\\
\Pr(\Delta|\Gamma_1) &= \text{Probability of getting the second or the third ball in the second draw, given that the first ball is obtained in the first draw } = 1,\\
\Pr(\Delta|\Gamma_2) &= \text{Probability of getting the second or the third ball in the second draw, given that the second ball is obtained in the first draw } = 1/2,\\
\Pr(\Delta|\Gamma_3) &= \text{Probability of getting the second or the third ball in the second draw, given that the third ball is obtained in the first draw } = 1/2.
\end{aligned}$$

The a posteriori probability $\Pr(\Gamma_1|\Delta)$ that the first ball *was* obtained in the first draw given that the ball obtained in the second draw is the second or the third one is calculated using the Bayes rule as:

$$\begin{aligned}
\Pr(\Gamma_1|\Delta) &= \frac{\Pr(\Delta|\Gamma_1)\Pr(\Gamma_1)}{\Pr(\Delta|\Gamma_1)\Pr(\Gamma_1)+\Pr(\Delta|\Gamma_2)\Pr(\Gamma_2)+\Pr(\Delta|\Gamma_3)\Pr(\Gamma_3)}\\
&= \frac{1\times\frac{1}{3}}{1\times\frac{1}{3}+\frac{1}{2}\times\frac{1}{3}+\frac{1}{2}\times\frac{1}{3}} = \frac{1}{2}.
\end{aligned}$$

One can similarly calculate $\Pr(\Gamma_2|\Delta) = \Pr(\Gamma_3|\Delta) = \frac{1}{4}$. This is expected, since the only events (x,y) consistent with $y \in \Delta$ are the four equiprobable possibilities $(1,2)$, $(1,3)$, $(2,3)$ and $(3,2)$. □

2.15.3 Expectation, Variance and Correlation

Let X be a random variable. The *expectation* $\mathrm{E}(X)$ of X is defined as follows:

$$\begin{aligned}
X \text{ is discrete:} &\qquad \mathrm{E}(X) := \sum_{x\in S_X} x\Pr(X=x) = \sum_{x\in S_X} x f_X(x).\\
X \text{ is continuous:} &\qquad \mathrm{E}(X) := \int_{-\infty}^{\infty} x f_X(x)\,\mathrm{d}x.
\end{aligned}$$

$\mathrm{E}(X)$ is also called the (*arithmetic*) *mean* or *average* of X. One uses the alternative symbols μ_X and $\overline{X}$ to denote $\mathrm{E}(X)$. More generally, let $X_1,\ldots,X_n$ be n random variables with joint probability distribution/density function $f(x_1,\ldots,x_n)$. Also let $g(X_1,\ldots,X_n) \in \mathbb{R}[X_1,\ldots,X_n]$. We define the following expectations:

X is discrete:

$$\mathrm{E}(g(X_1,\ldots,X_n)) := \sum_{x_1,\ldots,x_n} g(x_1,\ldots,x_n)f(x_1,\ldots,x_n),$$

X is continuous:

$$\mathrm{E}(g(X_1,\ldots,X_n)) := \int_{x_1=-\infty}^{\infty}\cdots\int_{x_n=-\infty}^{\infty} g(x_1,\ldots,x_n)f(x_1,\ldots,x_n)\,\mathrm{d}x_1\cdots\mathrm{d}x_n,$$

Let $g(X)$ and $h(Y)$ be real polynomial functions of the random variables X and Y and let $\alpha \in \mathbb{R}$. Then

$$\begin{aligned} \mathrm{E}(g(X)+h(Y)) &= \mathrm{E}(g(X))+\mathrm{E}(h(Y)), \\ \mathrm{E}(g(X)h(Y)) &= \mathrm{E}(g(X))\,\mathrm{E}(h(Y)) \text{ if } X \text{ and } Y \text{ are independent}, \\ \mathrm{E}(\alpha g(X)) &= \alpha\,\mathrm{E}(g(X)). \end{aligned}$$

Let us derive the sum and product formulas for *discrete* variables X and Y.

$$\begin{aligned} & \mathrm{E}(g(X)+h(Y)) \\ = & \sum_{x,y}[g(x)+h(y)]\Pr(X=x,Y=y) \\ = & \sum_{x,y} g(x)\Pr(X=x,Y=y)+\sum_{x,y} h(y)\Pr(X=x,Y=y) \\ = & \sum_{x}\left(g(x)\sum_{y}\Pr(X=x,Y=y)\right)+\sum_{y}\left(h(y)\sum_{x}\Pr(X=x,Y=y)\right) \\ = & \sum_{x}(g(x)\Pr(X=x))+\sum_{y}(h(y)\Pr(Y=y)) \\ = & \mathrm{E}(g(X))+\mathrm{E}(h(Y)). \end{aligned}$$

If X and Y are independent, then

$$\begin{aligned} & \mathrm{E}(g(X)h(Y)) \\ = & \sum_{x,y} g(x)h(y)\Pr(X=x,Y=y) \\ = & \sum_{x,y} g(x)h(y)\Pr(X=x)\Pr(Y=y) \\ = & \left(\sum_{x} g(x)\Pr(X=x)\right)\left(\sum_{y} h(y)\Pr(Y=y)\right) \\ = & \mathrm{E}(g(X))\,\mathrm{E}(h(Y)). \end{aligned}$$

The *variance* $\mathrm{Var}(X)$ of a random variable X is defined as

$$\mathrm{Var}(X) := \mathrm{E}\left[(X-\mathrm{E}(X))^2\right].$$

From the observation that $\mathrm{E}\left[(X-\mathrm{E}(X))^2\right] = \mathrm{E}\left[X^2-2\,\mathrm{E}(X)X+[E(X)]^2\right] = \mathrm{E}(X^2)-2\,\mathrm{E}(X)\,\mathrm{E}(X)+[E(X)]^2$, we derive the computational formula:

$$\mathrm{Var}(X) = \mathrm{E}[X^2]-[\mathrm{E}(X)]^2.$$

$\mathrm{Var}(X)$ is a measure of how the values of X are dispersed about the mean $\mathrm{E}(X)$ and is always a non-negative quantity. The (non-negative) square root of $\mathrm{Var}(X)$ is called the *standard deviation* σ_X of X:

$$\sigma_X := \sqrt{\mathrm{Var}(X)}.$$

The following formulas can be easily verified:

$$\begin{aligned}\mathrm{Var}(X+\alpha) &= \mathrm{Var}(X).\\ \mathrm{Var}(\alpha X) &= \alpha^2\,\mathrm{Var}(X).\\ \mathrm{Var}(X+Y) &= \mathrm{Var}(X)+\mathrm{Var}(Y)+2\,\mathrm{Cov}(X,Y),\end{aligned}$$

where $\alpha \in \mathbb{R}$ and where the *covariance* $\mathrm{Cov}(X,Y)$ of X and Y is defined as:

$$\mathrm{Cov}(X,Y) := \mathrm{E}\left[(X-\mathrm{E}(X))(Y-\mathrm{E}(Y))\right] = \mathrm{E}(XY)-\mathrm{E}(X)\,\mathrm{E}(Y)\,.$$

Normalized covariance is a measure of correlation between the two random variables X and Y. More precisely, the *correlation coefficient* $\rho_{X,Y}$ is defined as:

$$\rho_{X,Y} := \frac{\mathrm{Cov}(X,Y)}{\sqrt{\mathrm{Var}(X)\,\mathrm{Var}(Y)}} = \frac{\mathrm{Cov}(X,Y)}{\sigma_X\sigma_Y}\,.$$

If X and Y are independent, $\mathrm{E}(XY) = \mathrm{E}(X)\,\mathrm{E}(Y)$ so that $\mathrm{Cov}(X,Y) = 0$ and so $\rho_{X,Y} = 0$. The converse of this is, however, not true, that is, $\rho_{X,Y} = 0$ does not necessarily imply that X and Y are independent. $\rho_{X,Y}$ is a real value in the interval $[-1,1]$ and is a measure of *linear relationship* between X and Y. If larger (resp. smaller) values of X are (in general) associated with larger (resp. smaller) values of Y, then $\rho_{X,Y}$ is positive. On the other hand, if larger (resp. smaller) values of X are (in general) associated with smaller (resp. larger) values of Y, then $\rho_{X,Y}$ is negative.

Example 2.39 Once again consider the drawing of two balls from an urn containing three balls labelled $\{1,2,3\}$ (Examples 2.36, 2.37 and 2.38). Look at the second case (drawing without replacement). We use the shorthand notation P_{xy} for $\Pr(X=x, Y=y)$. The individual probability distributions of X and Y can be obtained from the joint distribution as follows:

$$\begin{aligned}\Pr(X=1) &= P_{11}+P_{12}+P_{13} = 0+(1/6)+(1/6) = 1/3\\ \Pr(X=2) &= P_{21}+P_{22}+P_{23} = (1/6)+0+(1/6) = 1/3\\ \Pr(X=3) &= P_{31}+P_{32}+P_{33} = (1/6)+(1/6)+0 = 1/3\end{aligned}$$

$$\begin{aligned}\Pr(Y=1) &= P_{11}+P_{21}+P_{31} = 0+(1/6)+(1/6) = 1/3\\ \Pr(Y=2) &= P_{12}+P_{22}+P_{32} = (1/6)+0+(1/6) = 1/3\\ \Pr(Y=3) &= P_{13}+P_{23}+P_{33} = (1/6)+(1/6)+0 = 1/3\end{aligned}$$

Thus $\mathrm{E}(X) = 1\times(1/3)+2\times(1/3)+3\times(1/3) = 2$. Similarly, $\mathrm{E}(Y) = 2$. Therefore, $\mathrm{E}(X+Y) = \mathrm{E}(X)+\mathrm{E}(Y) = 4$. This can also be verified by direct calculations: $\mathrm{E}(X+Y) = 3\times(1/3)+4\times(1/3)+5\times(1/3) = 4$.

$\mathrm{E}(X^2) = \mathrm{E}(Y^2) = 1^2\times(1/3)+2^2\times(1/3)+3^2\times(1/3) = 14/3$ and $\mathrm{Var}(X) = \mathrm{Var}(Y) = (14/3)-2^2 = 2/3$. The probability distribution for XY is

$$\begin{aligned}\mathrm{E}(XY=2) &= P_{12}+P_{21} = 1/3\\ \mathrm{E}(XY=3) &= P_{13}+P_{31} = 1/3\\ \mathrm{E}(XY=6) &= P_{23}+P_{32} = 1/3,\end{aligned}$$

so that $\mathrm{E}(XY) = 2 \times (1/3) + 3 \times (1/3) + 6 \times (1/3) = 11/3$. Therefore, $\mathrm{Cov}(XY) = \mathrm{E}(XY) - \mathrm{E}(X)\,\mathrm{E}(Y) = (11/3) - 2 \times 2 = -1/3$, that is,

$$\rho_{X,Y} = \mathrm{Cov}(X,Y)/\sqrt{\mathrm{Var}(X)\,\mathrm{Var}(Y)} = (-1/3)/(2/3) = -1/2.$$

The negative correlation between X and Y is expected. If $X = 1$ (small), Y takes bigger values $(2, 3)$. On the other hand, if $X = 3$ (large), Y assumes smaller values $(1, 2)$. Of course, the correlation is not perfect, since for $X = 2$ the values of Y can be smaller (1) or larger (3). So, we should feel happy to see a not-so-negative correlation of $-1/2$ between X and Y. □

2.15.4 Some Famous Probability Distributions

Some probability distributions that occur frequently in statistical theory and in practice are described now. Some other useful probability distributions are considered in the Exercises 2.169, 2.170 and 2.171.

Uniform distribution

A *discrete uniform random variable* U has sample space $S_U := \{x_1, \ldots, x_n\}$ and probability distribution

$$f_U(x_i) = \Pr(U = x_i) = \frac{1}{n} \quad \text{for all } i = 1, \ldots, n\,.$$

A *continuous uniform random variable* U has sample space S_U and probability density function

$$f_U(x) = \frac{1}{A} \quad \text{for all } x \in S_U\,,$$

where $A > 0$ is the size[23] of S_U. For example, if S_U is the real interval $[a, b]$ for $a < b$, we have

$$f_U(x) = \begin{cases} \frac{1}{b-a} & \text{if } a \leqslant x \leqslant b, \\ 0 & \text{otherwise.} \end{cases}$$

In this case, we have

$$\mathrm{E}(U) = (a+b)/2 \qquad \text{and} \qquad \mathrm{Var}(U) = (b-a)^2/12\,.$$

Uniform random variables often occur *naturally*. For example, if we throw an unbiased die, the six possible outcomes (1 through 6) are equally likely, that is, each possible outcome has the probability $1/6$. Similarly, if a real number is chosen randomly in the interval $[0, 1]$, we have a continuous uniform random variable. The built-in C library call `rand()` (pretends to) return an integer between 0 and $2^{31} - 1$, each with equal probability (namely, 2^{-31}).

[23] If $S_U \subseteq \mathbb{R}$, "size" means *length*. If $S_U \subseteq \mathbb{R}^2$ or $S_U \subseteq \mathbb{R}^3$, "size" refers to area or volume respectively. We assume that the size of S_U is "measurable".

Bernoulli distribution

The *Bernoulli random variable* $B = B(n,p)$ is a discrete random variable characterized by two parameters $n \in \mathbb{N}$ and $p \in [0,1]$, where p stands for the probability of a certain event E and n represents the number of (independent) trials. It is assumed that the probability of E remains constant (namely, p) in each of the n trials. The sample space $S_B = \{0, 1, \ldots, n\}$ comprises the (exact) numbers of occurrences of E in the n trials. B has the probability distribution

$$f_B(x) = \binom{n}{x} p^x (1-p)^{n-x} ,$$

as follows from simple combinatorial arguments. The mean and variance of B are:

$$\mathrm{E}(B) = np \qquad \text{and} \qquad \mathrm{Var}(B) = np(1-p) .$$

The Bernoulli distribution is also called the *binomial distribution*.

Normal distribution

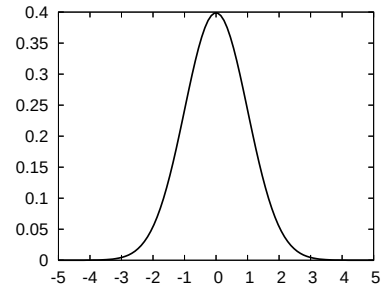
The *normal random variable* or the *Gaussian random variable* $N = N(\mu, \sigma^2)$ is a continuous random variable characterized by two real parameters μ and σ with $\sigma > 0$. The density function of N is

$$f_N(x) := \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/(2\sigma^2)}, \quad x \in \mathbb{R} .$$

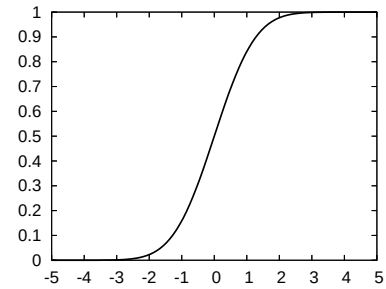
The cumulative distribution for N can be expressed in terms of the *error function* $\mathrm{erf}()$:

$$F_N(x) = \int_{-\infty}^{x} f_N(x)\,\mathrm{d}x = \frac{1}{2}\left[1 + \mathrm{erf}\left(\frac{x-\mu}{\sigma\sqrt{2}}\right)\right], \quad x \in \mathbb{R} .$$

The error function does not have a known closed-form expression. Figure 2.3 shows the curves for $f_N(x)$ and $F_N(x)$ for the parameter values $\mu = 0$ and $\sigma = 1$ (in this case, N is called the *standard normal variable*).



(a) $f(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$



(b) $F(x) = \frac{1}{2}\left[1 + \mathrm{erf}\left(\frac{x}{\sqrt{2}}\right)\right]$

Figure 2.3 Standard normal distribution

Some statistical properties of N are:

$$\mathrm{E}(N) = \mu \qquad \text{and} \qquad \mathrm{Var}(N) = \sigma^2 .$$

The curve $f_N(x)$ is symmetric about $x = \mu$. Most of the area under the curve is concentrated in the region $\mu - 3\sigma \leqslant x \leqslant \mu + 3\sigma$. More precisely:

$$\begin{aligned} \Pr(\mu - \sigma \leqslant X \leqslant \mu + \sigma) &\approx 0.68 , \\ \Pr(\mu - 2\sigma \leqslant X \leqslant \mu + 2\sigma) &\approx 0.95 , \\ \Pr(\mu - 3\sigma \leqslant X \leqslant \mu + 3\sigma) &\approx 0.997 . \end{aligned}$$

Many distributions occurring in practice (and in nature) approximately follow normal distributions. For example, the height of (adult) people in a given community is roughly normally distributed. Of course, the height of a person cannot be negative, whereas a normal random variable may assume negative values. But, in practice, the probability that such an approximating normal variable assumes a negative value is typically negligibly low.

2.15.5 Sample Mean, Variation and Correlation

In practice, we often do not know a priori the probability distribution or density function of a random variable X. In some cases, we do not have the complete data, whereas in some other cases we need an infinite amount of data to obtain the actual probability distribution of a random variable. For example, let X represent the life of an electric bulb manufactured by a given company in the last ten years. Even though there are only finitely many such bulbs and even if we assume that it is possible to trace the working of every such bulb, we have to wait until all these bulbs burn out, before we know the actual distribution of X. That is certainly impractical. On the contrary, if we have data on the life-times of some sample bulbs, we can approximate the properties of X by those of the samples.

Suppose that $S := (x_1, x_2, \ldots, x_n)$ is a sample of size n. We assume that all x_i are real numbers. We define the following quantities for S:

Mean $\bar{S} := \mu_S := \dfrac{1}{n} \sum_{i=1}^{n} x_i .$

Variance $\mathrm{Var}(S) := \dfrac{1}{n} \sum_{i=1}^{n} (x_i - \bar{S})^2 = \dfrac{1}{n} \left(\sum_{i=1}^{n} x_i^2 \right) - \bar{S}^2 = \overline{S^2} - \bar{S}^2 .$

Standard deviation $\sigma_S := \sqrt{\mathrm{Var}(S)} .$

Here $\overline{S^2}$ is the mean of the collection $S^2 := (x_1^2, x_2^2, \ldots, x_n^2)$.

If $T := (y_1, y_2, \ldots, y_m)$ is another sample (of real numbers), the (linear) relationship between S and T is measured by the following quantities:

Covariance $\mathrm{Cov}(S,T) := \dfrac{1}{nm} \sum_{i=1}^{n} \sum_{j=1}^{m} (x_i - \bar{S})(y_j - \bar{T}) = \overline{ST} - \bar{S}\bar{T} .$

Correlation $\rho_{S,T} := \dfrac{\mathrm{Cov}(S,T)}{\sqrt{\mathrm{Var}(S)\,\mathrm{Var}(T)}} = \dfrac{\mathrm{Cov}(S,T)}{\sigma_S \sigma_T} .$

Here $\overline{ST}$ is the mean of the collection $ST := (x_i y_j \mid i=1,\ldots,n,\ j=1,\ldots,m)$.

An important property of the normal distribution is the following:

Theorem 2.65 **Central limit theorem** Let X be *any* random variable with mean μ and variance σ^2 and let $n \in \mathbb{N}$. The mean $\bar{S}$ of a random sample S of size n chosen according to the distribution of X approximately follows the normal distribution $N(\mu, \sigma^2/n)$. The larger the sample size n is, the better this approximation is. ■

EXERCISE SET 2.15

2.162 An urn contains n_1 red balls and n_2 black balls. We draw k balls sequentially and randomly from the urn, where $1 \leqslant k \leqslant n_1 + n_2$.
(a) If the balls are drawn *with* replacement, what is the probability that the k-th ball drawn from the urn is red?
(b) If the balls are drawn *without* replacement, what is the probability that the k-th ball drawn from the urn is red?

2.163 Let X and Y be the random variables of Example 2.36. For each of the two cases, calculate the probability distribution functions, expectations and variances of the following random variables:
(a) XY
(b) $2X + 3Y$
(c) X^2
(d) $X^2 + 2XY + Y^2$
(e) $(X + Y)^2$

2.164 Let X and Y be *continuous* random variables, $g(X)$ and $h(Y)$ non-constant real polynomials and $\alpha, \beta, \gamma \in \mathbb{R}$. Prove that:

$$\begin{aligned}
\mathrm{E}(g(X) + h(Y)) &= \mathrm{E}(g(X)) + \mathrm{E}(h(Y)).\\
\mathrm{E}(g(X)h(Y)) &= \mathrm{E}(g(X))\,\mathrm{E}(h(Y)),\ \text{if } X \text{ and } Y \text{ are independent.}\\
\mathrm{E}(\alpha g(X)) &= \alpha\,\mathrm{E}(g(X)).\\
\mathrm{Var}(\alpha X + \beta Y + \gamma) &= \alpha^2\,\mathrm{Var}(X) + \beta^2\,\mathrm{Var}(Y).
\end{aligned}$$

2.165 Let X be a random variable and $Y := \alpha X + \beta$ for some $\alpha, \beta \in \mathbb{R}$. What is $\rho_{X,Y}$?

2.166 **(a)** Let X and Y be discrete random variables with joint probability distribution function $f(x, y)$. Show that the probability distributions of X and Y can be obtained as

$$\begin{aligned}
f_X(x) &= \sum_{y \in S_Y} f(x, y)\,,\\
f_Y(y) &= \sum_{x \in S_X} f(x, y)\,.
\end{aligned}$$

(b) If X and Y are continuous random variables with joint density function $f(x,y)$, show that the density functions of X and Y are given by

$$\begin{aligned} f_X(x) &= \int_{y=-\infty}^{+\infty} f(x,y)\,\mathrm{d}y\,, \\ f_Y(y) &= \int_{x=-\infty}^{+\infty} f(x,y)\,\mathrm{d}x\,. \end{aligned}$$

The functions f_X and f_Y are called the *marginal probability distribution* (or density function) of X and Y respectively.

2.167 Let X and Y be continuous random variables whose joint distribution is the uniform distribution in the triangle $0 \leqslant X \leqslant Y \leqslant 1$.
(a) Compute the marginal distributions f_X and f_Y.
(b) Compute $\mathrm{E}(X)$, $\mathrm{E}(Y)$, $\mathrm{Var}(X)$, $\mathrm{Var}(Y)$, $\mathrm{Cov}(X,Y)$ and $\rho_{X,Y}$.

2.168 Let X, Y, Z be random variables. Show that:

$$\begin{aligned} \mathrm{Cov}(X,Y) &= \mathrm{Cov}(Y,X)\,. \\ \rho_{X,Y} &= \rho_{Y,X}\,. \\ \mathrm{Cov}(X,X) &= \mathrm{Var}(X)\,. \\ \mathrm{Cov}(X,Y+Z) &= \mathrm{Cov}(X,Y)+\mathrm{Cov}(X,Z)\,. \\ \mathrm{Cov}(X,X+Y) &= \mathrm{Var}(X)+\mathrm{Cov}(X,Y)\,. \\ \mathrm{Cov}(X,X+Y) &= \mathrm{Var}(X) \quad \text{if } X \text{ and } Y \text{ are independent}\,. \end{aligned}$$

2.169 Geometric distribution Assume that in each trial of an experiment, an event E has a constant probability p of occurrence. Let $G = G(p)$ denote the random variable with $S_G = \mathbb{N}$ and with $f_G(x)$ equal to the probability that E occurs the first time during the x-th trial (that is, after exactly $x-1$ failures). Show that:

$$\begin{aligned} f_G(x) &= (1-p)^{x-1}p \quad \text{for each } x \in \mathbb{N}\,, \\ \mathrm{E}(G) &= \frac{1}{p} \quad (\text{if } p > 0)\,, \\ \mathrm{Var}(G) &= \frac{1-p}{p^2} \quad (\text{if } p > 0)\,. \end{aligned}$$

What if $p = 0$?

2.170 Poisson distribution Let $P = P(\lambda)$ be the discrete random variable with $S_P = \mathbb{N}_0$ and with $f_P(x) = e^{-\lambda}\frac{\lambda^x}{x!}$, where λ is a positive real constant. Show that $\mathrm{E}(P) = \mathrm{Var}(P) = \lambda$.

2.171 Exponential distribution
(a) Let $X = X(\lambda)$ be the continuous random variable with density

$$f_X(x) = \begin{cases} \lambda e^{-\lambda x} & \text{if } x \geqslant 0\,, \\ 0 & \text{if } x < 0\,, \end{cases}$$

where λ is a positive real constant. Show that:

$$\begin{aligned} F_X(x) &= \begin{cases} 1 - e^{-\lambda x} & \text{if } x \geqslant 0, \\ 0 & \text{if } x < 0, \end{cases} \\ \mathrm{E}(X) &= \frac{1}{\lambda}, \\ \mathrm{Var}(X) &= \frac{1}{\lambda^2}. \end{aligned}$$

(b) A random variable Y with $S_Y = \mathbb{R}$ is said to be *memoryless*, if

$$\Pr(Y > s + t \mid Y > s) = \Pr(Y > t) \quad \text{for all } s, t \in \mathbb{R}.$$

Show that the exponential variable X of Part (a) is memoryless.

2.172 The birthday paradox Let S be a finite set of cardinality n.
(a) Show that the probability that $k < n$ elements, drawn at random form S (with replacement), are (pairwise) distinct is

$$p := \frac{n(n-1)\cdots(n-k+1)}{n^k} = \prod_{i=1}^{k-1} \left(1 - \frac{i}{n}\right).$$

(b) Use the inequality $1 - x \leqslant e^{-x}$ for any real number x to show that $p \leqslant e^{\frac{-k(k-1)}{2n}}$.
(c) Deduce that $p \leqslant 1/2$, if $k \geqslant \frac{1}{2}\sqrt{1 + 8n \ln 2} \approx 1.18\sqrt{n}$, and that $p \leqslant 0.136$ for $k \geqslant 2\sqrt{n}$.
(The birthday paradox states that if only 23 people are chosen at random, there is a chance as high as 50 per cent that at least two of them have the same birthday.)

Chapter Summary

This chapter provides the foundations of public-key cryptology. The long compilation of mathematical concepts presented in the chapter would be indispensable for understanding the topics that follow in the next chapters.

This chapter begins with the basic concepts of *sets*, *functions* and *relations*. We also present the fundamental axioms of mathematics. Although the curricula of plus-two courses of many examination boards do include these topics, we planned to have a discussion on them in order to make our treatment self-sufficient.

Next comes a study of *groups* which are sets with binary operations satisfying some nice properties (associativity, identity, inverse and optionally commutativity). Groups are extremely important for cryptology. In particular, all discrete-log-based cryptosystems use suitable groups. *Subgroups*, *cosets* and formation of *quotient groups* constitute a prototypical feature that illustrates the basic paradigm of modern algebra. Secure cryptographic algorithms on groups rely on the availability of elements of large orders: for example, generators of big *cyclic groups*. We study these topics at length. Finally, we present *Sylow's theorem*. For us, this theorem has only theoretical significance; it is used for proving some other theorems.

A set with a single operation (like a group) is often too restrictive. Many mathematical structures we are familiar with (like integers, polynomials) are endowed with two basic operations addition and multiplication. A set with two such (compatible) operations is called a *ring*. A study of rings, *fields*, *ideals* and *quotient rings* is essential in algebra (and so in cryptography too). Three important types of rings, namely *unique factorization domains*, *principal ideal domains* and *Euclidean domains*, are also discussed. *Euclidean division* is an important property of integers and polynomials, and is useful from a computational perspective.

Then, as a specific example, we study the properties of $\mathbb{Z}$, the ring of integers. We concentrate mostly on elementary properties of integers like *divisibility*, *congruence*, *Chinese remainder theorem*, *Fermat's* and *Euler's theorems*, *quadratic residues* and the law of *quadratic reciprocity*. We finally discuss some assorted topics from analytic number theory. In cryptography, we require many big randomly generated primes. The *prime number theorem* guarantees that there is essentially an abundant source of primes. *Smooth integers* (that is, integers having only small prime divisors) are useful for modern algorithms that compute factorization and discrete logarithms. We present an estimate on the density of smooth integers. The last topic we study is the *Riemann hypothesis* and its generalizations. This yet unproven hypothesis has a bearing on the running times of many number-theoretic algorithms relevant to cryptology.

The next example is the ring of *polynomials* over a ring. Polynomials over a field admit Euclidean division and consequently unique factorization. *Irreducible polynomials* are useful for constructing *field extensions*. Extension fields of characteristic 2 are quite frequently used in cryptographic systems.

We subsequently study the theory of *vector spaces*. *Linear transformations* are appropriate maps between vector spaces and necessitate the theory of matrices. Matrix algebra is widely useful in cryptology as it is in any other branch of algorithmic computer science. Algorithms to solve linear systems over rings and fields constitute a basic computational tool. A study of *modules* and *algebras* at the end of this section is

mostly theoretical and can be avoided if the reader is willing to accept some theorems without proofs.

In the next section, we discuss the theory of *field extensions*. As mentioned earlier, cryptography relies heavily on extension fields of characteristic 2. Some related topics include *splitting fields* and *algebraic closure* of fields. At the end of this section, we have a short theoretical treatment of *Galois theory*.

Many popular cryptosystems are based on the multiplicative groups of *finite fields*. We study these fields as the next topic. Polynomials over finite fields are extremely useful for the construction and representation of finite fields. At the end of this section, we discuss several ways in which (elements of) finite fields can be represented in a computer's memory. This study expedites the design, analysis and efficient implementation of finite-field arithmetic.

Elliptic- and hyperelliptic-curve cryptography having gained popularity in recent years, one needs to study the theory of *plane algebraic curves*. This is what we do in the next three sections. To start with, we define *affine* and *projective spaces* and *curves*. Going from the affine space to the projective space is necessitated by a systematic (algebraic) inclusion of *points at infinity* on a plane curve. We also discuss the theory of *divisors* and the *Jacobian* on plane curves. For *elliptic curves*, the Jacobian can be replaced by the equivalent group described in terms of the *chord and tangent rule*. For *hyperelliptic curves*, on the other hand, we have little option other than understanding the Jacobian itself.

Two kinds of elliptic curves that must be avoided in cryptography are *supersingular* curves and *anomalous curves*. The *elliptic curve group* (over a finite field) is the basic set used in elliptic curve cryptosystems. The orders (cardinality) of these groups are given by *Hasse's theorem*. The *structure theorem* establishes that an elliptic curve group (over a finite field) is not necessarily cyclic, but has a rank of at most two.

We then study *Jacobians* of *hyperelliptic curves* over finite fields. This study supplements the theory of divisors on general curves. *Reduced* and *semi-reduced divisors* are expedient for the representation of the elements in the Jacobian of a hyperelliptic curve.

Many popular cryptosystems (including RSA) derive their security (presumably) from the intractability of the integer factorization problem. The best algorithm known till date for factoring integers is the number-field sieve method. An understanding of this algorithm requires the knowledge of *number fields* and *number rings*. We devote a section to the study of these mathematical objects. We start with some necessary *commutative algebra* including *localization*, *integral dependence* and *Noetherian rings*. Next, we deal with *Dedekind domains*. All number rings are Dedekind domains in which ideals admit *unique factorization*. We also discuss the factorization of ideals in number rings generated by rational primes and the structure of units in number rings (*Dirichlet's unit theorem*).

The next section is a gentle introduction to the theory of p*-adic numbers*. These numbers are useful, for example, for designing attacks against elliptic curve cryptosystems.

In the last section, we summarize some statistical tools. Under the assumption that the reader is already familiar with the elementary notion of probability, we discuss properties of *random variables* and of some common *probability distributions* (including *uniform* and *normal distributions*). The *birthday paradox* described in an

exercise is often useful in cryptographic context (for example, for collision attacks on hash functions).

That is the end of this chapter. The compilation may initially look long and boring, perhaps intimidating too. The unfortunate reality is that public-key cryptology is mathematical, and it is arguably better to treat it in the formal way. If the reader is not comfortable with mathematics (in general), cryptology is perhaps not her cup of tea. An elementary approach to cryptology is what many other books have adopted. This book aims at being different in that respect. It is up to the reader to decide to what level of details she is willing to study cryptography.

Suggestions for Further Reading

Knowledge is of two kinds. We know a subject ourselves, or we know where we can find information upon it.
— Samuel Johnson

In this chapter, we have summarized the basic mathematical facts that cryptologists are expected to know in order to have a decent understanding of the present-day public-key technology. Our discussion has been often more intuitive than mathematically complete. A reader willing to gain further insight in these areas should look at materials written specifically to deal with the specialized topics. Here are our (biased) suggestions.

There are numerous textbooks on introductory algebra. The books by Herstein [125], Fraleigh [96], Dummit and Foote [81], Hungerford [133] and Adkins and Weintraub [1] are some of our favourites. The algebra of commutative rings with identity (rings by our definition) is called *commutative algebra* and is the basic for learning advanced areas of mathematics like algebraic geometry and algebraic number theory. A serious study of these disciplines demands more in-depth knowledge of commutative algebra than we have presented in Section 2.13.1. Atiyah and MacDonald's book [14] is a de facto standard on commutative algebra. Hoffman and Kunze's book [127] is a good reference for linear algebra and matrix algebra.

Elementary number theory deals with the theory of (natural) numbers without using sophisticated techniques from complex analysis and algebra. Zuckerman et al. [316] can be consulted for a lucid introduction to this subject. The books by Burton [42] and Mollin [207] are good alternatives.

Thorough mathematical treatise on finite fields can be found in the books by Lidl and Niederreiter [179, 180] of which the second also deals with computational issues. Other books of computational flavour include those by Menezes [191] and by Shparlinski [274]. Also see the paper [273] by Shparlinski.

The use of elliptic curves in cryptography has been proposed by Koblitz [150] and Miller [205], and that of hyperelliptic curves by Koblitz [151]. A fair mathematical understanding of elliptic curves banks on the knowledge of commutative algebra (see above) and algebraic geometry. Hartshorne's book [124] is a detailed introduction to algebraic geometry. Fulton's book [99] on algebraic curves is another good reference. Rigorous mathematical treatment on elliptic curves can be found in Silverman's books [275, 276]. The book by Koblitz [152] is elementary, but has a somewhat different focus than needed in cryptology. By far, the best short-cut is the recent textbook from Washington [298]. Some other books by Koblitz [150, 153, 154], Blake et al. [24], Menezes [192] and Hankerson et al. [123] are written for non-experts in algebraic geometry (and hence lack mathematical details), but are good from computational viewpoint. The expository reports [46, 47] by Charlap et al. provide nice *elementary* introduction to elliptic curves. For hyperelliptic curves, on the other hand, no such books are available. Koblitz's book [154] includes a chapter on hyperelliptic curves. In addition, an appendix in the same book, written by Menezes et al. much in the style of Charlap et al. [46, 47], provides an introductory and elementary coverage.

In an oversimplified sense, algebraic number theory deals with the study of number fields. The books by Janusz [140], Lang [160], Mollin [208] and Ribenboim [251] go

well beyond what we cover in Section 2.13. Also see [89]. For a more modern and sophisticated treatment, look at Neukirch's book [216]. A book dedicated to p-adic numbers is due to Koblitz [149]. Course notes from one of the authors of *this* book can also be useful in this regard. The notes are freely downloadable from:

http://www.facweb.iitkgp.ernet.in/∼adas/IITK/course/MTH617/SS02/

Analytic number theory deals with the application of complex analytic techniques to solve problems in number theory. Although we do not explicitly need this branch of mathematics (apart from a few theorems that we mention without proofs), it is rather important for the study of numbers. Consult the books by Apostol [12] and by Ireland and Rosen [136] for this. Also see [249]. For complex analysis, we recommend the book by Ahlfors [6]

Feller's celebrated book [92] is a classical reference on probability theory. Grinstead and Snell's book [121] is available in the Internet.

Algebraic and Number-theoretic Computations

3

3.1 Introduction
3.2 Complexity Issues
3.3 Multiple-precision Integer Arithmetic
3.4 Elementary Number-theoretic Computations
3.5 Arithmetic in Finite Fields
3.6 Arithmetic on Elliptic Curves
3.7 Arithmetic on Hyperelliptic Curves
3.8 Random Numbers
Chapter Summary
Sugestions for Further Reading

From the start there has been a curious affinity between mathematics, mind and computing . . . It is perhaps no accident that Pascal and Leibniz in the seventeenth century, Babbage and George Boole in the nineteenth, and Alan Turing and John von Neumann in the twentieth – seminal figures in the history of computing – were all, among their other accomplishments, mathematicians, possessing a natural affinity for symbol, representation, abstraction and logic.
— Doron Swade [295]

. . . the laws of physics and of logic . . . the number system . . . the principle of algebraic substitution. These are ghosts. We just believe in them so thoroughly they seem real.
— Robert M. Pirsig [233]

The world is continuous, but the mind is discrete.
— David Mumford

3.1 Introduction

Now that we have studied the properties of important mathematical objects that play vital roles in public-key cryptology, it is time to concentrate on the algorithmic and implementation issues for working with these objects. We need well-defined schemes (data structures) to represent these objects and well-defined procedures (algorithms) to manipulate them. While a theoretical analysis of the performance of our data structures and algorithms is of great concern, it still leaves us in the abstract domain. In the long run, one has to translate the abstract statements in the algorithms to machine codes that the computer understands, and this is where the implementation tidbits come into picture. It is our personal experience that a naive implementation of an algorithm may run hundred times slower than a carefully optimized implementation of the same algorithm. In certain specific applications (like those based on smart cards), where memory is a scarce resource, one should also pay attention to the storage requirements of the data structures and code segments. This chapter is an introduction to all these specialized topics.

Before we proceed further, certain comments are in order. In this book, we describe algorithms using a *pseudocode* that closely resembles the syntax of the programming language C. The biggest difference between C and our pseudocode is that we have given preference to mathematical notations in place of C syntax. For example, $=$ means equality in our codes, whereas assignment is denoted by $:=$. Similarly, our while and for loops look more human-readable, for example, for $i = 0, 1, \ldots, m-1$ instead of C's for (i=0; i<m; i++). In order to understand our pseudocode, a knowledge of C (or a similar programming language) is helpful, but not essential, on the part of the reader.

For certain implementations, we assume that the target machine carries out 32-bit 2's-complement arithmetic. This is indeed true for most modern PCs and personal work stations. By the term *word*, we mean a 32-bit unit in the computer memory. We will also assume that the compiler provides facilities for storing and doing arithmetic with unsigned 64-bit integers. Though this is *not* an ANSI C feature, most popular compilers used today *do* support this built-in data type (Examples: unsigned __int64 for the Microsoft Visual C++ Compiler and unsigned long long for the GNU C Compiler). Though it is apparently desirable to be more generic and to avoid these specific assumptions on the part of the machine and the compiler, our exposition highlights the power of fine-tuning based on the knowledge of the underlying system.

3.2 Complexity Issues

Given an algorithm (or an implementation of the same), the time and space required for the execution of the algorithm on a machine depend very much on the machine's architecture and on the compiler. But this does not mean that we cannot make some general theoretical estimates. The so-called asymptotic estimates that we are going to introduce now tend to approach the real situation as the input size tends to infinity. For *finite* input sizes (which is always the case in practice), these theoretical predictions turn out to provide valuable guidelines.

3.2.1 Order Notations

We start with the following important definitions.

Definition 3.1 Let f and g be positive real-valued functions of natural numbers.

(1) f is said to be *bounded above by* g or *of the order of* g, denoted $f = \mathrm{O}(g)$, if there exists an $n_0 \in \mathbb{N}$ and a positive real constant c such that $f(n) \leqslant cg(n)$ for all $n \geqslant n_0$. In this case, we also say that g is *bounded below by* f and denote this by $g = \Omega(f)$.

(2) If $f = \mathrm{O}(g)$ and $g = \mathrm{O}(f)$, we say that f and g are *of the same order* and denote this by $f = \Theta(g)$ (or by $g = \Theta(f)$). Equivalently, $f = \Theta(g)$ if and only if $f = \mathrm{O}(g)$ and $f = \Omega(g)$; that is, if and only if there exist an integer $n_0 \in \mathbb{N}$ and real positive constants c_1, c_2 such that $c_1 g(n) \leqslant f(n) \leqslant c_2 g(n)$ for all $n \geqslant n_0$.

(3) f is said to be of *strictly lower order* than g, denoted $f = \mathrm{o}(g)$, if $f(n)/g(n)$ tends to 0 as n tends to infinity. In other words, $f = \mathrm{o}(g)$ if and only if for every real positive constant c (however small it may be) there exists an integer $n_c \in \mathbb{N}$ such that $f(n) < cg(n)$ for all $n \geqslant n_c$. If $f = \mathrm{o}(g)$, we also say that g is of *strictly higher order* than f and denote this by $g = \omega(f)$. Thus $g = \omega(f)$ if and only if for every real positive constant c (however large it may be) there exists an integer $n_c \in \mathbb{N}$ such that $g(n) > cf(n)$ for all $n \geqslant n_c$. □

Example 3.1

(1) Let $f(n) := a_d n^d + \cdots + a_1 n + a_0$ with $d \geqslant 0$, $a_i \in \mathbb{R}$, $a_d > 0$. Then $f = \Theta(n^d)$. This heuristically means that as n becomes sufficiently large, the leading term $a_d n^d$ dominates over the other terms, and apart from the constant of proportionality a_d the function $f(n)$ grows with n as n^d does. If $f = \Theta(n^d)$ for some integer $d > 0$, we say that f is of *polynomial order* in n.[1] A $\Theta(1)$ function is often called a *constant function*.

(2) If $f = \Theta((\log n)^a)$ for some real $a > 0$, we say that f is of *poly-logarithmic order* in n. By Exercise 3.2(b), any function of poly-logarithmic order grows asymptotically slower than any function of polynomial order.

(3) If $f = \Theta(a^n)$ for some real $a > 1$, f said to be of *exponential order* in n. Again by Exercise 3.2(b) any function of exponential order grows asymptotically faster than any function of polynomial order.

(4) Now, consider a function of the form

$$f(n) = \exp\left(c\, n^{\alpha} (\ln n)^{1-\alpha}\right) \tag{3.1}$$

for real $c > 0$ and for $0 \leqslant \alpha \leqslant 1$. For $\alpha = 0$, we have $f = \Theta(n^c)$; that is, f is of polynomial order. On the other extreme, if $\alpha = 1$, $f = \Theta(a^n)$, where $a := \exp(c)$, that is, f is of exponential order. If $0 < \alpha < 1$, we say that f

[1] This is not the complete truth. Functions like $\sqrt{n}$, $n^{2.3}$ or $n^3(\log n)^2$ would be better included in the *polynomial* family. Thus, we may define f to be of polynomial order (in n), if $f = \mathrm{O}(n^d)$ and $f = \Omega(n^{d'})$ for some positive *real* constants d, d'. Similar comments hold for poly-logarithmic and exponential orders.

is of *subexponential order* in n, since the order of f is somewhere in between polynomial and exponential. We will come across functions of subexponential orders quite frequently in the rest of the book. Note that as α increases from 0 to 1, the order of f also *increases* monotonically from polynomial to exponential.

(5) A function $f = \mathrm{O}(n^a(\log n)^b)$ with $a > 0$ and $b \geqslant 0$ is often denoted by the *soft* O*-notation*: $f = \tilde{\mathrm{O}}(n^a)$. This implies that up to multiplication by a polynomial in $\log n$ the function f is of the order of n^a. Similarly, if $f = \mathrm{O}(a^n g(n))$ for $a > 1$ and for some $g(n)$ of polynomial order, we say that $f = \tilde{\mathrm{O}}(a^n)$. Intuitively spoken, the O-notation hides constant multipliers, whereas the soft O-notation suppresses exponentially small multipliers.

(6) The notion of order can be readily extended to functions with two or more input variables. For example, for positive real-valued functions f, g of two positive integer variables m, n one says $f = \mathrm{O}(g)$, if for some $m_0, n_0 \in \mathbb{N}$ and for some positive real constant c one has $f(m,n) \leqslant cg(m,n)$ for all $m \geqslant m_0$ and $n \geqslant n_0$. The function $f(m,n) = m^3 2^n$ is of polynomial order in m, but of exponential order in n. □

The order notation is used to analyse algorithms in the following way. For an algorithm, the *input size* is defined as the total number of bits needed to represent the input of the algorithm. We find asymptotic estimates of the running time and the memory requirement of the algorithm in terms of its input size. Let $f(n)$ denote the running time[2] of an algorithm A for an input of size $n \in \mathbb{N}$. If $f(n) = \Theta(n^a)$ (or, more generally, if $f = \mathrm{O}(n^a)$) for some $a > 0$, A is called a *polynomial-time algorithm*. If $a = 1$ (resp. $2, 3, \ldots$), then A is specifically called a *linear-time* (resp. *quadratic-time*, *cubic-time*, ...) algorithm. A $\Theta(1)$ algorithm is often called a *constant-time* algorithm. If $f = \Theta(b^n)$ for some $b > 1$, A is called an *exponential-time algorithm*. Similarly, if f satisfies Equation (3.1) with $0 < \alpha < 1$, A is called a *subexponential-time algorithm*.

One has similar classifications of an algorithm in terms of its space requirements, namely, *polynomial-space*, *linear-space*, *exponential-space*, and so on. We can afford to be lazy and drop *-time* from the adjectives introduced in the previous paragraph. Thus, an exponential algorithm is an exponential-time algorithm, not an exponential-space algorithm.

It is expedient to note here that the running time of an algorithm may depend on the particular instance of the input, even when the input size is kept fixed. For an example, see Exercise 3.3. We should, therefore, be prepared to distinguish, for a given algorithm and for a given input size n, between the *best* (that is, shortest) running time $f_b(n)$, the *worst* (that is, longest) running time $f_w(n)$, the *average* running time $f_a(n)$ on all possible inputs (of size n) and the *expected* running time $f_e(n)$ for a randomly chosen input (of size n). In typical situations, $f_w(n)$, $f_a(n)$ and $f_e(n)$ are of the same order, in which case we simply denote, by *running time*, one of these functions. If this is not the case, an unqualified use of the phrase *running time* would denote the *worst* running time $f_w(n)$.

[2]The practical running time of an algorithm may vary widely depending on its implementation and also on the processor, the compiler and even on run-time conditions. Since we are talking about the order of growth of running times in relation to the input size, we neglect the constants of proportionality and so these variations are usually not a problem. If one plans to be more concrete, one may measure the running time by the number of bit operations needed by the algorithm.

The order notation, though apparently attractive and useful, has certain drawbacks. First it depicts the behaviour of functions (like running times) as the input size tends to infinity. In practice, one always has finite input sizes. One can check that if $f(n) = n^{100}$ and $g(n) = (1.01)^n$ are the running times of two algorithms A and B respectively (for solving the same problem), then $f(n) \leqslant g(n)$ if and only if $n = 1$ or $n \geqslant 117{,}309$. But then if the input size is only $1{,}000$, one would prefer the exponential-time algorithm B over the polynomial-time algorithm A. Thus asymptotic estimates need not guarantee correct suggestions at practical ranges of interest. On the other hand, an algorithm which is a product of human intellect does not tend to have such extreme values for the parameters; that is, in a polynomial-time algorithm, the degree is usually $\leqslant 10$ and the base for an exponential-time algorithm is usually not as close to 1 as 1.01 is. If we have $f(n) = n^5$ and $g(n) = 2^n$ as the respective running times of the algorithms A and B, then A outperforms B (in terms of speed) for all $n \geqslant 23$.

The second drawback of the order notation is that it suppresses the constant of proportionality; that is, an algorithm whose running time is $100n^2$ has the same order as one whose running time is n^2. This is, however, a situation that we cannot neglect in practice. In particular, when we compare two different implementations of the same algorithm, the one with a smaller constant of proportionality is more desirable than the one with a larger constant. This is where implementation tricks prove to be important and even indispensable for large-scale applications.

3.2.2 Randomized Algorithms

A *deterministic algorithm* is one that always follows the same sequence of computations (and thereby produces the same output) for a given input. The *deterministic running time* of a computational problem P is the fastest of the running times (in order notation) of the *known* algorithms to solve P.

If an algorithm makes some random choices during execution, we call the algorithm *randomized* or *probabilistic*. The exact sequence of computations followed by the algorithm depends on these random choices and as a result different executions of the same algorithm may produce different outputs for a given input. At first glance, randomized algorithms look useless, because getting different outputs for a given input is apparently not what one would really want. But there are situations where this is desirable. For example, in an implementation of the RSA protocol, one generates random primes p and q of given bit lengths. Here we require our prime generation procedure to produce different primes during different executions (that is, for different entities on the net).

More importantly, randomized algorithms often provide *practical* computational solutions for many problems for which no practical deterministic algorithms are known. We will shortly encounter many such situations where randomized algorithms are simplest and/or fastest known algorithms. However, this sudden enhancement in performance by random choices does not come for free. To explain the so-called darker sides of randomization, we explain two different types of randomized algorithms.

A *Monte Carlo algorithm* is a randomized algorithm that may produce *incorrect* outputs. However, for such an algorithm to be useful, we require that the running time be always small and the probability of an error sufficiently low. A good example of a Monte Carlo algorithm is Miller–Rabin's algorithm (Algorithm 3.13) for testing

the primality of an integer. For an integer of bit size n, the Miller–Rabin test with t iterations runs in time $\mathrm{O}(tn^3)$. Whenever the algorithm outputs *false*, it is always correct. But an answer of *true* is incorrect with an error probability $\leqslant 2^{-2t}$, that is, it certifies a composite integer as a prime with probability $\leqslant 2^{-2t}$. For $t = 20$, an error is expected to occur less than once in every 10^{12} executions. With this little sacrifice we achieve a running time of $\mathrm{O}(n^3)$ (for a fixed t), whereas the best deterministic primality testing algorithm (known to the authors at the time of writing this book) takes time $\mathrm{O}(n^{7.5})$ and hence is not practical.

A *Las Vegas algorithm* is a randomized algorithm which always produces the correct output. However the running time of such an algorithm depends on the random choices made. For such an algorithm to be useful, we expect that for *most* random choices the running time is small. As an example, consider the problem of finding a random (monic) irreducible polynomial of degree n over $\mathbb{F}_2$. Algorithm 3.22 tests the irreducibility of a polynomial in $\mathbb{F}_2[X]$ in deterministic polynomial time. We generate random polynomials of degree n and check the irreducibility of these polynomials by Algorithm 3.22. From Section 2.9.2, we know that a randomly chosen monic polynomial of degree n over a finite field is irreducible with an approximate probability of $1/n$. This implies that after $\mathrm{O}(n)$ random polynomials are tried, one *expects* to find an irreducible polynomial. The resulting Las Vegas algorithm (Algorithm 3.23) runs in expected polynomial time. It may, however, happen that for certain random choices we keep on generating reducible polynomials for an exponential number of times, but the likelihood of such an accident is very, very low (Exercise 3.5).

An algorithm is said to be a *probabilistic* or *randomized polynomial-time algorithm*, if it is either a Monte Carlo algorithm with polynomial *worst* running time or a Las Vegas algorithm with polynomial *expected* running time. Both the above examples of randomized algorithms are probabilistic polynomial-time algorithms. A combination of these two types of algorithms can also be conceived; namely, algorithms that produce correct outputs with high probability *and* have polynomial expected running time. Some computational problems are so challenging that even such *probably correct and probably fast* algorithms are quite welcome.

We finally note that there are certain computational problems for which the deterministic running time is exponential and for which randomization also does not help much. In some cases, we have subexponential randomized algorithms which are still too slow to be of reasonable practical use. Some of these so-called *intractable* problems are at the heart of the security of many public-key cryptographic protocols.

3.2.3 Reduction Between Computational Problems

In the last two sections, we have introduced theoretical measures (the order notations) for estimating the (known) difficulty of solving computational problems. In this section, we introduce another concept by which we can compare the relative difficulty of two computational problems.

Let P_1 and P_2 be two computational problems. We say that P_1 is *polynomial-time reducible* to P_2 and denote this as $P_1 \preceq P_2$, if there is a polynomial-time algorithm which, given a solution of P_2, provides a solution for P_1. This means that if $P_1 \preceq P_2$, then the problem P_1 is no more difficult than P_2 apart from the extra polynomial-time reduction effort. In that case, if we know an algorithm to solve P_2 in polynomial time,

then we have a polynomial-time algorithm for P_1 too. If $P_1 \preceq P_2$ and $P_2 \preceq P_1$, we say that the problems P_1 and P_2 are *polynomial-time equivalent* and write $P_1 \cong P_2$.

In order to give an example of these concepts, we let G be a finite cyclic multiplicative group of order n and g a generator of G. The *discrete logarithm problem* (*DLP*) is the problem of computing for a given $a \in G$ an integer x such that $a = g^x$. The *Diffie–Hellman problem* (*DHP*), on the other hand, is the problem of computing g^{xy} from the given values of g^x and g^y. If one can compute y from g^y, one can also compute $g^{xy} = (g^x)^y$ by performing an exponentiation in the group G. Therefore, DHP $\preceq$ DLP, if exponentiations in G can be computed in polynomial time. In other words, if a solution for DLP is known, a solution for DHP is also available: that is, DHP is no more difficult than DLP except for the additional exponentiation effort. However, the reverse implication (that is, whether DLP $\preceq$ DHP) is not known for many groups.

So far we have assumed that our reduction algorithms are deterministic. If we allow randomized (that is, probabilistic) polynomial-time reduction algorithms, we can similarly introduce the concepts of *randomized polynomial-time reducibility* and of *randomized polynomial-time equivalence*. We urge the reader to formulate the formal definitions for these concepts.

EXERCISE SET 3.2

3.1 **(a)** Sort the following functions in the increasing sequence of order. (Don't mind if some of these functions are not defined for a few values of n.)

$$10^{12}, 2^n, 2^{2^n}, 2^{n^2}, 100n^2, 10^{-3}n^3, 1/n, 1/\sqrt{n}, n!, n^n,$$
$$\log n, (\log n)/n, n/\log n, n^2 \log n, n(\log n)^2, (0.1)^{\log n}, (\log n)^n,$$
$$1/\log n, 2^{\sqrt{\log n}}, 10^6(\log n)^{100}, \log\log n, 2^{\log\log n}, n^{\log\log n},$$
$$\exp(\sqrt{n \ln n}), \exp(\sqrt{n \lg n}), \exp(\sqrt{\ln n \ln\ln n}),$$
$$\exp(n^{1/3}(\ln n)^{2/3}), \exp((\ln n)^{1/3}(\ln\ln n)^{2/3}).$$

(b) Evaluate the functions of Part (a) at $n = 10^i$ for $i = 1, 2, \ldots, 10$ and conclude that as n gets larger, the asymptotic ordering tallies with the actual ordering more correctly.

3.2 **(a)** Show that for any real $a > 1$ and $b > 0$ one has $n^b = \mathrm{o}(a^n)$.
(b) For any positive real c, d, show that $(\log n)^c = \mathrm{o}(n^d)$.
(c) Show that if $f = \mathrm{O}(g)$ and $g = \mathrm{O}(h)$, then $f = \mathrm{O}(h)$.
(d) Give an example to show that $f = \mathrm{O}(g)$ does not necessarily imply $f = \Theta(g)$.
(e) Give an example of a function f with $f = \mathrm{O}(n^{1+\epsilon})$ for every $\epsilon > 0$, but f is not $\mathrm{O}(n)$.

3.3 Suppose that an algorithm A takes as input a bit string and runs in time $g(t)$, where t is the number of one-bits in the input string. Let $f_b(n)$, $f_w(n)$, $f_a(n)$ and $f_e(n)$ respectively denote the best, worst, average and expected running times of A for inputs of size n. Derive the following table under the assumption that each of the 2^n bit strings of length n is equally likely.

	Running times			
$g(t)$	$f_b(n)$	$f_w(n)$	$f_a(n)$	$f_e(n)$
t	0	n	$n/2$	$n/2$
t^2	0	n^2	$n(n+1)/4$	$n^2/4$
2^t	1	2^n	$(3/2)^n$	$(\sqrt{2})^n$

3.4 **(a)** Show that an exponential-space (resp. subexponential-space) algorithm must be (at least) exponential-time (resp. subexponential-time) too. You may assume that at a time a computing device can access (read/write) at most a finite number of memory locations.
(b) Give an example of an algorithm that is exponential-time but polynomial-space.

3.5 Consider the Las Vegas algorithm discussed in Section 3.2.2 for generating a random irreducible polynomial of degree n over $\mathbb{F}_2$. Assume that a randomly chosen polynomial in $\mathbb{F}_2[X]$ of degree n has (an exact) probability of $1/n$ for being irreducible. Find out the probability p_r that r polynomials chosen randomly (with repetition) from $\mathbb{F}_2[X]$ are all reducible. For $n = 1000$, calculate the numerical values of p_r for $r = 10^i$, $i = 1, \ldots, 6$, and find the smallest integers r for which $p_r \leqslant 1/2$ and $p_r \leqslant 10^{-12}$. Find the expected number of polynomials tested for irreducibility, before the algorithm terminates.

3.6 Let $n = pq$ be the product of two distinct primes p and q. Show that factoring n is polynomial-time equivalent to computing $\phi(n) = (p-1)(q-1)$, where ϕ is Euler's totient function. (Assume that an arithmetic operation (including computation of integer square roots) on integers of bit size t can be performed in polynomial time (in t).)

3.7 Let G be a finite cyclic multiplicative group and let H be the subgroup of G generated by $h \in G$ whose order is *known*. The *generalized discrete logarithm problem* (*GDLP*) is the following: Given $a \in G$, find out if $a \in H$ and, if so, find an integer x for which $a = h^x$. Show that GDLP $\cong$ DLP, if exponentiations in G can be carried out in polynomial time and if DLP in H is polynomial-time equivalent to DLP in G. **[H]**

3.3 Multiple-precision Integer Arithmetic

Cryptographic protocols based on the rings $\mathbb{Z}_n$ and $\mathbb{F}_p$ demand n and p to be sufficiently large (of bit length $\geqslant 512$) in order to achieve the desired level of security. However, standard compilers do not support data types to hold with *full precision* the integers of this size. For example, C compilers support integers of size $\leqslant 64$ bits. So one must employ custom-designed data types for representing and working with such big integers. Many libraries are already available that can handle integers of arbitrary length. FREELIP, GMP, LiDIA, NTL and ZEN are some such libraries that are even freely available.

Alternatively, one may design one's own functions for multiple-precision integers. Such a programming exercise is not very difficult, but making the functions run *efficiently* is a huge challenge. Several tricks and optimization techniques can turn a naive implementation to a much faster and more memory-efficient code and it needs years of experimental experience to find out the subtleties. Theoretical asymptotic estimates might serve as a guideline, but only experimentation can settle the relative merits and demerits of the available algorithms for input sizes of practical interest. For example, the theoretically fastest algorithm known for multiplying two multiple-precision integers is based on the so-called *fast Fourier transform* (*FFT*) techniques. But our experience shows that this algorithm starts to outperform other common but asymptotically slower algorithms only when the input size is at least several thousand bits. Since such very large integers are rarely needed by cryptographic protocols, FFT-based multiplication is not useful in this context.

3.3.1 Representation of Large Integers

In order to represent a large integer, we break it up into small parts and store each part in a memory word[3] accessible by built-in data types. The simplest way to break up a (positive) integer a is to predetermine a *radix* $\mathfrak{R}$ and compute the $\mathfrak{R}$-ary representation $(a_{s-1}, \ldots, a_0)_{\mathfrak{R}}$ of a (see Exercise 3.8). One should have $\mathfrak{R} \leqslant 2^{32}$ so that each $\mathfrak{R}$-ary digit a_i can be stored in a memory word. For the sake of efficiency, it is advisable to take $\mathfrak{R}$ to be a power of 2. It is also expedient to take $\mathfrak{R}$ as large as possible, because smaller values of $\mathfrak{R}$ lead to (possibly) longer size s and thereby add to the storage requirement and also to the running time of arithmetic functions. The best choice is $\mathfrak{R} = 2^{32}$. We denote by ulong a built-in unsigned integer data type provided by the compiler (like the ANSI C standard unsigned long). We use an array of ulong for storing the digits. The array can be static or dynamic. Though dynamic arrays are more storage-efficient (because they can be allocated only as much memory as needed), they have memory allocation and deallocation overheads and are somewhat more complicated to programme than static arrays. Moreover, for cryptographic protocols one typically needs integers no longer than 4096 bits. Since the product of two integers of bit size t has bit size $\leqslant 2t$, a static array of $8192/32 = 256$ ulong suffices for storing cryptographic integers. It is also necessary to keep track of the actual size of an integer, since filling up with leading 0 digits is not an efficient strategy. Finally, it is often useful to have a signed representation of integers. A sign bit is also necessary for this case. We state three possible declarations in Exercise 3.11.

3.3.2 Basic Arithmetic Operations

We now describe the implementations of addition, subtraction, multiplication and Euclidean division of multiple-precision integers. Every other complex operation (like modular arithmetic, gcd) is based on these primitives. It is, therefore, of utmost importance to write efficient codes for these basic operations.

For integers of cryptographic sizes, the most efficient algorithms are the standard ones we use for doing arithmetic on decimal numbers, that is, for two positive integers

[3] We assume that a word in the memory is 32 bits long.

$a = a_{s-1} \dots a_0$ and $b = b_{t-1} \dots b_0$ we compute the sum $c = a + b = c_{r-1} \dots c_0$ as follows. We first compute $a_0 + b_0$. If this sum is $\geqslant \mathfrak{R}$, then $c_0 = a_0 + b_0 - \mathfrak{R}$ and the carry is 1, otherwise $c_0 = a_0 + b_0$ and the carry is 0. We then compute $a_1 + b_1$ plus the carry available from the previous digit, and compute c_1 and the next carry as before.

For computing the product $d = ab = d_{l-1} \dots d_0$, we do the usual quadratic procedure; namely, we initialize all the digits of d to 0 and for each $i = 0, \dots, s-1$ and $j = 0, \dots, t-1$ we compute $a_i b_j$ and add it to the $(i+j)$-th digit of d. If this sum (call it σ) at the $(i+j)$-th location exceeds $\mathfrak{R} - 1$, we find out q, r with $\sigma = q\mathfrak{R} + r$, $r < \mathfrak{R}$. Then d_{i+j} is assigned r, and q is added to the $(i+j+1)$-st location. If that addition results in a carry, we propagate the carry to higher locations until it gets fully absorbed in some word of d.

All these sound simple, but complications arise when we consider the fact that the sum of two 32-bit words (and a possible carry from the previous location) may be 33 bits long. For multiplication, the situation is even worse, because the product $a_i b_j$ can be 64 bits long. Since our machine word can hold only 32 bits, it becomes problematic to hold all these intermediate sums and products to full precision. We assume that the least significant 32 bits are correctly returned and assigned to the output variable (ulong), whereas the leading 32 bits are lost.[4] The most efficient way to keep track of these overflows is to use assembly instructions and this is what many number theory packages (like PARI and UBASIC) do. But this means that for every target architecture we have to write different assembly codes. Here we describe certain tricks that make it possible to grab the overflow information with only high-level languages, without sufficiently degrading the performance compared to assembly instructions.

Addition and subtraction

First consider the sum $a_i + b_i$. We compute the least significant 32 bits by assigning $c_i = a_i + b_i$. It is easy to see that an overflow occurs during this sum if and only if $c_i < a_i$. We set the output carry accordingly. Now, let us consider the situation when we have an input carry: that is, when we compute the sum $c_i = a_i + b_i + 1$. Here an overflow occurs if and only if $c_i \leqslant a_i$. Algorithm 3.1 performs this addition of words.

Algorithm 3.1 **Addition of words**

Input: Words a_i and b_i and the input carry $\gamma_i \in \{0, 1\}$.
Output: Word c_i and the output carry $\delta_i \in \{0, 1\}$ with $a_i + b_i + \gamma_i = c_i + \delta_i \mathfrak{R}$.
Steps:
$c_i := a_i + b_i$.
if (γ_i) { $c_i{+}{+}$, $\delta_i :=$ ($(c_i \leqslant a_i)$? 1 : 0). } else { $\delta_i :=$ ($(c_i < a_i)$? 1 : 0). }

Algorithm 3.1 assumes that c_i and a_i are stored in different memory words. If this is not the case, we should store $a_i + b_i$ in a temporary variable and, after the second line, c_i should be assigned the value of this temporary variable. Note also that many processors provide an increment primitive which is faster than the general addition primitive. In that case, the statement $c_i{+}{+}$ is preferable to $c_i := c_i + 1$.

[4] This is the typical behaviour of a CPU that supports 2's complement arithmetic.

For subtraction, we proceed analogously from right to left and keep track of the borrow. Here the check for overflow can be done before the subtraction of words is carried out (and, therefore, no temporary variable is needed, if we assume that the output carry is not stored in the location of the operands).

Algorithm 3.2 **Subtraction of words**

Input: Words a_i and b_i and the input borrow $\gamma_i \in \{0, 1\}$.
Output: Word c_i and the output borrow $\delta_i \in \{0, 1\}$ with $a_i - b_i - \gamma_i = c_i - \delta_i \mathfrak{R}$.
Steps:
if (γ_i) { $\delta_i :=$ ($(a_i \leqslant b_i)$? $1 : 0$), $c_i := a_i - b_i$, $c_i --$. }
else { $\delta_i :=$ ($(a_i < b_i)$? $1 : 0$), $c_i := a_i - b_i$. }

We urge the reader to develop the complete addition and subtraction procedures for multiple-precision integers, based on the above primitives for words.

Multiplication

The product of two 32-bit words can be as long as 64 bits, and we plan to (compute and) store this product in two words. Assuming the availability of a built-in 64-bit unsigned integer data type (which we will henceforth denote as ullong), this can be performed as in Algorithm 3.3.

Algorithm 3.3 **Multiplication of words**

Input: Words a and b.
Output: Words c and d with $ab = c\mathfrak{R} + d$.
Steps:
/* We use a temporary variable t of data type ullong */
$t :=$ (ullong)$(a) *$ (ullong)(b), $c :=$ (ulong)$(t \gg 32)$, $d :=$ (ulong)t.

We use a temporary 64-bit integer variable t to store the product ab. The lower 32 bits of t is stored in d by simply typecasting, whereas the higher 32 bits of t is obtained by right-shifting t (the operator $\gg$) by 32 bits. This is a reasonable strategy given that we do not explore assembly-level instructions. Algorithm 3.4 describes a multiplication algorithm for two multiple-precision integer operands, that does not directly use the word-multiplying primitive of Algorithm 3.3.

The reader can verify easily that this code properly computes the product. We now highlight how this makes the computation efficient. The intermediate results are stored in the array t of 64-bit ullong. This means that after the 64-bit product $a_i b_j$ of words a_i and b_j is computed (in the temporary variable T), we directly add T to the location t_{i+j}. If the sum exceeds $\mathfrak{R}^2 - 1 = 2^{64} - 1$, that is, if an overflow occurs, we should add $\mathfrak{R}$ to t_{i+j+1} or equivalently 1 to t_{i+j+2}. This last addition is one of ullong integers and can be made more efficient, if this is replaced by ulong increments, and this is what we do using the temporary array u. Since the quadratic loop is the bottleneck of

Algorithm 3.4 **Multiplication of multiple-precision integers**

Input: Integers $a = (a_{r-1} \dots a_0)_{\mathfrak{R}}$ and $b = (b_{s-1} \dots b_0)_{\mathfrak{R}}$
Output: The product $c = (c_{r+s-1} \dots c_0)_{\mathfrak{R}} = ab$.
Steps:
/* Let T be a variable and $t_0, \dots, t_{r+s-1}$ an array of ullong variables */
/* Let v be a variable and $u_0, \dots, u_{r+s-1}$ an array of ulong variables */

Initialize the array locations c_i, t_i and u_i to 0 for all $i = 0, \dots, r+s-1$.

/* The quadratic loop */
for ($i = 0, \dots, r-1$) and ($j = 0, \dots, s-1$) {
  $T :=$ (ullong)$(a_i) * $ (ullong)(b_j).
  if $((t_{i+j} \mathrel{+}= T) < T)$ $u_{i+j+2}{+}{+}$.
}

/* Deferred normalization */
for ($i = 0, \dots, r+s-1$) {
  if $((c_i \mathrel{+}= u_i) < u_i)$ $u_{i+1}{+}{+}$.
  $v :=$ (ulong)(t_i), if $((c_i \mathrel{+}= v) < v)$ $u_{i+1}{+}{+}$.
  $v :=$ (ulong)$(t_i \gg 32)$, if $((c_{i+1} \mathrel{+}= v) < v)$ $u_{i+2}{+}{+}$.
}

the multiplication procedure, it is absolutely necessary to make this loop as efficient as possible.

After the quadratic loop, we do deferred normalization from the array of 64-bit double-words t_i to the array of 32-bit words c_i. This is done using the typecasting and right-shift strategy mentioned in Algorithm 3.3. We should also take care of the intermediate carries stored in the array u. The normalization loop takes a total time of $\mathrm{O}(r+s)$, whereas the quadratic loop takes time $\mathrm{O}(rs)$. If we had done normalization inside the quadratic loop itself, that would incur an additional $\mathrm{O}(rs)$ cost (which is significantly more than that of deferred normalization).

Squaring

If both the operands a and b of multiplication are same, it is not necessary to compute $a_i b_j$ and $a_j b_i$ separately. We should add to t_{i+j} the product $a_i a_j = a_i^2$, if $i = j$, or the product $2a_i a_j$, if $i < j$. Note that $2a_i a_j$ can be computed by left shifting $a_i a_j$ by one bit. This might result in an overflow which can be checked before shifting by looking at the 64th bit of $a_i a_j$. Algorithm 3.5 incorporates these changes.

Fast multiplication

For the multiplication of two multiple-precision integers, there are algorithms that are asymptotically faster than the quadratic Algorithms 3.4 and 3.5. However, not all these theoretically faster algorithms are practical for sizes of integers used in cryptology. Our

Algorithm 3.5 **The quadratic loop for squaring**

```
for (i = 0, ..., r − 1) and (j = i, ..., r − 1) {
  T := (ullong)(a_i) ∗ (ullong)(a_j).
  if (i ≠ j) {
    if (the 64th bit of T is 1) u_{i+j+2}++.
    T ≪= 1.
  }
  if ((t_{i+j} += T) < T) u_{i+j+2}++.
}
```

practical experience shows that a strategy due to *Karatsuba* outperforms the quadratic algorithm, if both the operands are of roughly equal sizes and if the bit lengths of the operands are 300 or more. We describe Karatsuba's algorithm in connection with squaring, where the two operands are same (and hence of the same size). Suppose we want to compute a^2 for a multiple-precision integer $a = (a_{r-1} \dots a_0)_{\mathfrak{R}}$. We first break a into two integers of almost equal sizes, namely, $\alpha := (a_{r-1} \dots a_t)_{\mathfrak{R}}$ and $\beta := (a_{t-1} \dots a_0)_{\mathfrak{R}}$, so that $a = \mathfrak{R}^t\alpha + \beta$. Now, $a^2 = \alpha^2\mathfrak{R}^{2t} + 2\alpha\beta\mathfrak{R}^t + \beta^2$ and $2\alpha\beta = (\alpha^2 + \beta^2) - (\alpha - \beta)^2$. We recursively invoke Karatsuba's multiplication with operands α, β and $\alpha - \beta$. Recursion continues as long as the operands are not too small and the depth of recursion is within a prescribed limit. One can check that Karatsuba's algorithm runs in time $\mathrm{O}(r^{\lg 3} \lg r) = \mathrm{O}(r^{1.585} \lg r)$ which is a definite improvement over the $\mathrm{O}(r^2)$ running time taken by the quadratic algorithm.

The best-known algorithm for multiplication of two multiple-precision integers is based on the fast Fourier transform (FFT) techniques and has running time $\mathrm{O}\tilde{}(r)$. However, for integers used in cryptology this algorithm is usually not practical. Therefore, we will not discuss FFT multiplication in this book.

Division

Euclidean division with remainder of multiple-precision integers is somewhat cumbersome, although conceptually as difficult (that is, as simple) as the division procedure of decimal integers, taught in early days of school. The most challenging part in the procedure is *guessing* the next digit in the quotient. For decimal integers, we usually do this by looking at the first few (decimal) digits of the divisor and the dividend. This need not give us the correct digit, but something *close* to the same. In the case of $\mathfrak{R}$-ary digits, we also make a guess of the quotient digit based on a few leading $\mathfrak{R}$-ary digits of the divisor and the dividend, but certain precautions have to be taken to ensure that the guess is not *too different* from the correct one.

Suppose we are given positive integers $a = (a_{r-1} \dots a_0)_{\mathfrak{R}}$ and $b = (b_{s-1} \dots b_0)_{\mathfrak{R}}$ with $a_{r-1} \neq 0$ and $b_{s-1} \neq 0$, and we want to compute the integers $x = (x_{r-s} \dots x_0)_{\mathfrak{R}}$ and $y = (y_{s-1} \dots y_0)_{\mathfrak{R}}$ with $a = xb + y$, $0 \leqslant y < b$. First, we want that $b_{s-1} \geqslant \mathfrak{R}/2$ (you'll see why, later). If this condition is already not met, we force it by multiplying both a and b by 2^t for some suitable t, $0 < t < 32$. In that case, the quotient remains the same, but the remainder gets multiplied by 2^t. The desired remainder can be later

Algorithm 3.6 **Euclidean division of multiple-precision integers**

Input: Integers $a = (a_{r-1} \dots a_0)_{\mathfrak{R}}$ and $b = (b_{s-1} \dots b_0)_{\mathfrak{R}}$ with $r \geqslant 3$, $s \geqslant 2$, $a_{r-1} \neq 0$, $b_{s-1} \geqslant \mathfrak{R}/2$ and $a \geqslant b$.
Output: The quotient $x = (x_{r-s} \dots x_0)_{\mathfrak{R}} = a \text{ quot } b$ and the remainder $y = (y_{s-1} \dots y_0)_{\mathfrak{R}} = a \text{ rem } b$ of Euclidean division of a by b.
Steps:
Initialize the quotient digits x_i to 0 for $i = 0, \dots, r-s$.

```
/* The main loop */
for (i = r − 1, ..., s) {
   /* Initial check */
   if (a_i ⩾ b_{s−1}) and (a ⩾ b𝔑^{i−s+1}) { x_{i−s+1}++, a := a − b𝔑^{i−s+1}. }

   /* Guess the next digit of quotient */
   if (a_i = b_{s−1}) x_{i−s} := 𝔑 − 1, else x_{i−s} := ⌊(a_i𝔑 + a_{i−1})/b_{s−1})⌋.
   if (x_{i−s} ≠ 0)
      while (x_{i−s}(b_{s−1}𝔑 + b_{s−2}) > a_i𝔑^2 + a_{i−1}𝔑 + a_{i−2}) x_{i−s}−−.

   /* Modify the guess to the correct value */
   z := x_{i−s}b𝔑^{i−s}.
   if (a < z) { x_{i−s}−−, z := z − b𝔑^{i−s}. }
   a := a − z.
}

/* Here the quotient may be one less than the actual value */
if (a ⩾ b) { a := a − b, x := x + 1. }
y := a.
```

found out easily by right-shifting the computed remainder by t bits. The process of making $b_{s-1} \geqslant \mathfrak{R}/2$ is often called *normalization* (of b). Henceforth, we will assume that b is normalized. Note that normalization may increase the word-size of a by 1.

Algorithm 3.6 implements multiple-precision division. It is not difficult to prove the correctness of the algorithm. We refrain from doing so, but make some useful comments. The initial check inside the main loop may cause the increment of x_{i-s+1}. This may lead to a carry which has to be adjusted to higher digits. This carry propagation is not mentioned in the code for simplicity. Since b is assumed to be normalized, this initial check needs to be carried out only once; that is, for a non-normalized b we have to replace the if statement by a while loop. This is the first advantage of normalization. In the first step of guessing the quotient digit x_{i-s}, we compute $\lfloor (a_i\mathfrak{R} + a_{i-1})/b_{s-1} \rfloor$ using ullong arithmetic. At this point, the guess is based only on two leading digits of a and one leading digit of b. In the while loop, we refine this guess by considering one more digit of a and b each. Since b is normalized, this while loop is executed no more than twice (the second advantage of normalization). The guess for x_{i-s} made in this way is either equal to or one more than the correct value which is then computed by comparing a with $x_{i-s}b\mathfrak{R}^{i-s}$. The running time of the algorithm is $\mathrm{O}(s(r-s))$. For a fixed r, this is maximum (namely $\mathrm{O}(r^2)$) when $s \approx r/2$.

Bit-wise operations

Multiplication and division by a power of 2 can be carried out more efficiently using bit operations (on words) instead of calling the general procedures just described. It is also often necessary to compute the bit length of a non-zero multiple-precision integer and the multiplicity of 2 in it. In these cases also, one should use bit operations for efficiency. For these implementations, it is advantageous to maintain precomputed tables of the constants 2^i, $i = 0, \ldots, 31$, and of $2^i - 1$, $i = 0, \ldots, 32$, rather than computing them *in situ* every time they are needed. In Algorithm 3.7, we describe an implementation of multiplication by a power of 2 (that is, the *left shift* operation). We use the symbols OR, $\gg$ and $\ll$ to denote bit-wise *or*, *right shift* and *left shift* operations on 32-bit integers.

Algorithm 3.7 **Left-shift of multiple-precision integers**

Input: Integer $a = (a_{r-1} \ldots a_0)_{\mathfrak{R}} \neq 0$, $a_{r-1} \neq 0$, and $t \in \mathbb{N}$.
Output: The integer $c = (c_{s-1} \ldots c_0)_{\mathfrak{R}} = a \cdot 2^t$, $c_{s-1} \neq 0$.
Steps:
$u := t \operatorname{quot} 32$, $v := t \operatorname{rem} 32$.
if $(v = 0)$ { /* Word-by-word copy */
  $s := r + u$.
  for $(i = r - 1, \ldots, 0)$ $c_{i+u} := a_i$.
} else { /* Use shifts of individual words */
  $s := r + u + 1$, $c_{s-1} := 0$.
  for $(i = r-1, \ldots, 0)$ { $c_{i+u+1} := c_{i+u+1} \operatorname{OR} (a_i \gg (32-v))$, $c_{i+u} := (a_i \ll v)$. }
  if $(c_{s-1} = 0)$ $s--$.
}
for $(i = u - 1, \ldots, 0)$ $c_i := 0$.

Unless otherwise mentioned, we will henceforth forget about the above structural representation of multiple-precision integers and denote arithmetic operations on them by the standard symbols ($+$, $-$, $*$ or $\cdot$ or $\times$, quot, rem and so on).

3.3.3 GCD

Computing the greatest common divisor of two (multiple-precision) integers has important applications. In this section, we assume that we want to compute the (positive) gcd of two positive integers a and b. The Euclidean gcd loop comprising repeated division (Proposition 2.15) is not usually the most efficient way to compute integer gcds. We describe the *binary gcd* algorithm that turns out to be faster for practical bit sizes of the operands a and b. If $a = 2^r a'$ and $b = 2^s b'$ with a' and b' odd, then $\gcd(a, b) = 2^{\min(r,s)} \gcd(a', b')$. Therefore, we may assume that a and b are odd. In that case, if $a > b$, then $\gcd(a, b) = \gcd(a - b, b) = \gcd((a - b)/2^t, b)$, where $t := \mathrm{v}_2(a - b)$ is the multiplicity of 2 in $a - b$. Since the sum of the bit sizes of $(a-b)/2^t$ and b is strictly smaller than that of a and b, repeating the above computation eventually terminates the algorithm after finitely many iterations.

Algorithm 3.8 **Extended binary gcd**

Input: Two positive integers a, b with $a \geqslant b$ and b odd.
Output: Integers d, u and v with $d = \gcd(a, b) = ua + vb > 0$. If $(a, b) \neq (1, 1)$, then $|u| < b$ and $|v| < a$.
Steps:

```
/* Initial reduction */
Compute integers q and r satisfying a = bq + r with 0 ⩽ r < b.
if (r = 0) { (d, u, v) := (b, 0, 1), return. }

/* Initialize */
(x, y) := (b, r).
v1 := 0, v2 := 1.

/* Main loop */
while (1) {
  if (x ⩾ y) {
    x := x − y.    /* x is even here except perhaps in the first iteration */
    v1 := v1 − v2.
    if (x = 0) {    /* End loop and return d, u and v */
      u2 := (y − v2 r)/b.
      (d, u, v) := (y, v2, u2 − v2 q).
      Return.
    } else if (x is even) {
      t := v2(x), x := x/2^t.    /* x is odd here */
      for (i = 1, ..., t) {
        if (v1 is odd)  v1 := v1 + b.
        v1 := v1/2.
      }
    }
  } else { /* if (x < y) */
    y := y − x, v2 := v2 − v1.    /* y is even here */
    t := v2(y), y := y/2^t.    /* y is odd here */
    for (i = 1, ..., t) {
      if (v2 is odd)  v2 := v2 + b.
      v2 := v2/2.
    }
  }
}
```

Multiple-precision division is much costlier than subtraction followed by division by a power of 2. This is why the binary gcd algorithm outperforms the Euclidean gcd algorithm. However, if the bit sizes of a and b differ reasonably, it is preferable to use Euclidean division once and replace the pair (a, b) by $(b, a \bmod b)$, before entering the binary gcd loop. Even when the original bit sizes of a and b are not much different, one may carry out this initial reduction, because in this case Euclidean division does not take much time.

Recall from Proposition 2.16 that if $d := \gcd(a, b)$, then for some integers u and v we have $d = ua + vb$. Computation of d along with a pair of integers u, v is called

the *extended gcd computation*. Both the Euclidean and the binary gcd loops can be augmented to compute these integers u and v. Since binary gcd is faster than Euclidean gcd, we describe an implementation of the extended binary gcd algorithm. We assume that $0 < b \leqslant a$ and compute u and v in such a way that if $(a, b) \neq (1, 1)$, then $|u| < b$ and $|v| < a$. Algorithm 3.8, which shows the details, requires b to be odd. The other operand a may also be odd, though the working of the algorithm does not require this.

In order to prove the correctness of Algorithm 3.8, we introduce the sequence of integers x_k, y_k, $u_{1,k}$, $u_{2,k}$, $v_{1,k}$ and $v_{2,k}$ for $k = 0, 1, 2, \ldots$, initialized as:

$$\begin{aligned} x_0 &:= b, \quad u_{1,0} := 1, \quad v_{1,0} := 0, \\ y_0 &:= r, \quad u_{2,0} := 0, \quad v_{2,0} := 1. \end{aligned}$$

During the k-th iteration of the main loop, $k = 1, 2, \ldots$, we modify the values x_{k-1}, y_{k-1}, $u_{1,k-1}$, $u_{2,k-1}$, $v_{1,k-1}$ and $v_{2,k-1}$ to x_k, y_k, $u_{1,k}$, $u_{2,k}$, $v_{1,k}$ and $v_{2,k}$ in such a way that we always maintain the relations:

$$\begin{aligned} u_{1,k}x_0 + v_{1,k}y_0 &= x_k, \\ u_{2,k}x_0 + v_{2,k}y_0 &= y_k. \end{aligned}$$

The main loop terminates when $x_k = 0$, and at that point we have the desired relation $y_k = \gcd(b, r) = u_{2,k}b + v_{2,k}r$. For the updating during the k-th iteration, we assume that $x_{k-1} \geqslant y_{k-1}$. (The converse inequality can be handled analogously.) The x and y values are updated as $x_k := (x_{k-1} - y_{k-1})/2^{t_k}$, $y_k := y_{k-1}$, where $t_k := \mathrm{v}_2(x_{k-1} - y_{k-1})$. Thus, we have $u_{2,k} = u_{2,k-1}$ and $v_{2,k} = v_{2,k-1}$, whereas if $t_k > 0$, we write

$$\begin{aligned} x_k &= \frac{x_{k-1} - y_{k-1}}{2^{t_k}} = \frac{(u_{1,k-1} - u_{2,k-1})x_0 + (v_{1,k-1} - v_{2,k-1})y_0}{2^{t_k}} \\ &= \begin{cases} \dfrac{\left[\frac{u_{1,k-1}-u_{2,k-1}}{2}\right]x_0 + \left[\frac{v_{1,k-1}-v_{2,k-1}}{2}\right]y_0}{2^{t_k-1}} & \text{if } v_{1,k-1} - v_{2,k-1} \text{ is even,} \\ \dfrac{\left[\frac{u_{1,k-1}-u_{2,k-1}-y_0}{2}\right]x_0 + \left[\frac{v_{1,k-1}-v_{2,k-1}+x_0}{2}\right]y_0}{2^{t_k-1}} & \text{if } v_{1,k-1} - v_{2,k-1} \text{ is odd.} \end{cases} \end{aligned}$$

All the expressions within square brackets in the last equation are integers, since $x_0 = b$ is odd. Note that updating the variables in the loop requires only the values of these variables available from the previous iteration. Therefore, we may drop the prefix k and call these variables x, y, u_1, u_2, v_1 and v_2. Moreover, the variables u_1 and u_2 need not be maintained and updated in every iteration, since the updating procedure for the other variables does not depend on the values of u_1 and u_2. We need the value of u_2 only at the end of the main loop, and this is available from the relation $y = u_2b + v_2r$ maintained throughout the loop. The formula $u_2b + v_2r = y = \gcd(b, r)$ is then combined with the relations $a = qb + r$ and $\gcd(a, b) = \gcd(b, r)$ to get the final relation $\gcd(a, b) = v_2a + (u_2 - v_2q)b$.

Algorithm 3.8 continues to work even when $a < b$, but in that case the initial reduction simply interchanges a and b and we forfeit the possibility of the reduction in size of the arguments (x and y) caused by the initial Euclidean division.

Finally, we remove the restriction that b is odd. We write $a = 2^r a'$ and $b = 2^s b'$ with a', b' odd and call Algorithm 3.8 with a' and b' as parameters (swapping a' and b', if $a' < b'$) to compute integers d', u', v' with $d' = \gcd(a', b') = u'a' + v'b'$. Without

loss of generality, assume that $r \geqslant s$. Then $d := \gcd(a, b) = 2^s d' = u'(2^s a') + v'b$. If $r = s$, then $2^s a' = a$ and we are done. So assume that $r > s$. If u' is even, we can extract a power of 2 from u' and multiply $2^s a'$ by this power. So let's say that we have a situation of the form $d = \bar{u}(2^t a') + \bar{v}b$ for some integers $\bar{u}$ and $\bar{v}$, with $\bar{u}$ odd, and for $s \leqslant t < r$. We can rewrite this as $d = (\bar{u}+b')(2^t a')+(\bar{v}-2^{t-s}a')b$. Since $\bar{u}+b'$ is even, this gives us $d = \tilde{u}(2^\tau a') + \tilde{v}b$, where $\tau > t$ and where $\tilde{u}$ is odd or $\tau = r$. Proceeding in this way, we eventually reach a relation of the form $d = u(2^r a') + vb = ua + vb$. It is easy to check that if $(a', b') \neq (1, 1)$, then the integers u and v obtained as above satisfy $|u| < b$ and $|v| < a$.

3.3.4 Modular Arithmetic

So far, we have described how we can represent and work with the elements of $\mathbb{Z}$. In cryptology, we are seemingly more interested in the arithmetic of the rings $\mathbb{Z}_n$ for multiple-precision integers n. We canonically represent the elements of $\mathbb{Z}_n$ by integers between 0 and $n - 1$.

Let $a, b \in \mathbb{Z}_n = \{0, 1, \ldots, n-1\}$. In order to compute $a+b$ in $\mathbb{Z}_n$, we compute the integer sum $a+b$, and, if $a+b \geqslant n$, we subtract n from $a+b$. This gives us the desired canonical representative in $\mathbb{Z}_n$. Similarly, for computing $a-b$ in $\mathbb{Z}_n$, we subtract b from a as integers, and, if the difference is negative, we add n to it. For computing $ab \in \mathbb{Z}_n$, we multiply a and b as integers and then take the remainder of Euclidean division of this product by n.

Note that $a \in \mathbb{Z}_n$ is invertible (that is, $a \in \mathbb{Z}_n^*$) if and only if $\gcd(a, n) = 1$. For $a \in \mathbb{Z}_n$, $a \neq 0$, we call the extended (binary) gcd algorithm with a and n as the arguments and get integers d, u, v satisfying $d = \gcd(a, n) = ua+vn$. If $d > 1$, a is not invertible modulo n. Otherwise, we have $ua \equiv 1 \pmod{n}$, that is, $a^{-1} \equiv u \pmod{n}$. The extended gcd algorithm indeed returns a value of u satisfying $|u| < n$. Thus if $u > 0$, it is the canonical representative of a^{-1}, whereas if $u < 0$, then $u + n$ is the canonical representative of a^{-1}.

Modular exponentiation

Another frequently needed operation in $\mathbb{Z}_n$ is *modular exponentiation*, that is, the computation of a^e for some $a \in \mathbb{Z}_n$ and $e \in \mathbb{Z}$. Since $a^0 = 1$ for all $a \in \mathbb{Z}_n$ and since $a^e = (a^{-1})^{-e}$ for $e < 0$ and $a \in \mathbb{Z}_n^*$, we may assume, without loss of generality, that $e \in \mathbb{N}$. Computing the integral power a^e followed by taking the remainder of Euclidean division by n is not an efficient way to compute a^e in $\mathbb{Z}_n$. Instead, after every multiplication, we reduce the product modulo n. This keeps the size of the intermediate products small. Furthermore, it is also a bad idea to compute a^e as $(\cdots((a\cdot a)\cdot a)\cdots a)$ which involves $e-1$ multiplications. It is possible to compute a^e using $\mathrm{O}(\lg e)$ multiplications and $\mathrm{O}(\lg e)$ squarings in $\mathbb{Z}_n$, as Algorithm 3.9 suggests. This algorithm requires the bits of the binary expansion of the exponent e, which are easily obtained by bit operations on the words of e.

The for loop iteratively computes $b_i := a^{(e_{r-1}\ldots e_i)_2} \pmod{n}$ starting from the initial value $b_r := 1$. Since $(e_{r-1} \ldots e_i)_2 = 2(e_{r-1} \ldots e_{i+1})_2 + e_i$, we have $b_i \equiv b_{i+1}^2 a^{e_i} \pmod{n}$. This establishes the correctness of the algorithm. The squaring

Algorithm 3.9 **Modular exponentiation: square-and-multiply algorithm**

Input: $a \in \mathbb{Z}_n$, $e \in \mathbb{N}$.
Output: $b = a^e \in \mathbb{Z}_n$.
Steps:
Let the binary expansion of e be $e = (e_{r-1} \dots e_1 e_0)_2$, where each $e_i \in \{0, 1\}$.
$b := 1$.
for ($i = r-1, \dots, 0$) {
  $b := b^2 \pmod n$. /* Squaring */
  if ($e_i = 1$) $b := ba \pmod n$. /* Multiplication */
}

(b^2) and multiplication (ba) inside the for loop of the algorithm are computed in $\mathbb{Z}_n$ (that is, as integer multiplication followed by reduction modulo n). If we assume that $e_{r-1} = 1$, then $r = \lceil \lg e \rceil$. The algorithm carries out r squares and $\rho \leqslant r$ multiplications in $\mathbb{Z}_n$, where ρ is the number of bits of e, that are 1. On an average $\rho = r/2$. Algorithm 3.9 runs in time $\mathrm{O}((\log e)(\log n)^2)$. Typically, $e = \mathrm{O}(n)$, so this running time is $\mathrm{O}((\log n)^3)$.

Now, we describe a simple variant of this square-and-multiply algorithm, in which we choose a small t and use the 2^t-ary representation of the exponent e. The case $t = 1$ corresponds to Algorithm 3.9. In practical situations, $t = 4$ is a good choice. As in Algorithm 3.9, multiplication and squaring are done in $\mathbb{Z}_n$.

Algorithm 3.10 **Modular exponentiation: windowed square-and-multiply algorithm**

Input: $a \in \mathbb{Z}_n$, $e \in \mathbb{N}$.
Output: $b = a^e \in \mathbb{Z}_n$.
Steps:
Let $e = (e_{r-1} \dots e_1 e_0)_{2^t}$, where each $e_i \in \{0, 1, \dots, 2^t - 1\}$.
Compute and store $a^l \in \mathbb{Z}_n$ for $l = 0, 1, \dots, 2^t - 1$. /* Precomputation */
$b := 1$.
for ($i = r-1, \dots, 0$) {
  for ($j = 1, \dots, t$) $b := b^2 \pmod n$. /* Squaring */
  $b := ba^{e_i} \pmod n$. /* Multiplication: Read a^{e_i} from the precomputed table */
}

In Algorithm 3.10, the powers a^l, $l = 0, 1, \dots, 2^t - 1$, are precomputed using the formulas: $a^0 = 1$, $a^1 = a$ and $a^l = a^{l-1} \cdot a$ for $l \geqslant 2$. The number of squares inside the for loop remains (almost) the same as in Algorithm 3.9. However, the number of multiplications in this loop reduces at the expense of the precomputation step. For example, let n be an integer of bit length 1024 and let $e \approx n$. A randomly chosen e of this size has about 512 one-bits. Therefore, the for loop of Algorithm 3.9 does about 512 multiplications, whereas with $t = 4$ Algorithm 3.10 does only $1024/4 = 256$ multiplications with the precomputation step requiring 14 multiplications. Thus, the total number of multiplications reduces from (about) 512 to $14 + 256 = 270$.

Montgomery exponentiation

During a modular exponentiation in $\mathbb{Z}_n$, every reduction (computation of remainder) is done by the fixed modulus n. *Montgomery exponentiation* exploits this fact and speeds up each modular reduction at the cost of some preprocessing overhead.

Assume that the storage of n requires s $\mathfrak{R}$-ary digits, that is, $n = (n_{s-1} \dots n_0)_{\mathfrak{R}}$ (with $n_{s-1} \neq 0$). Take $R := \mathfrak{R}^s = 2^{32s}$, so that $R > n$. As is typical in most cryptographic situations, n is an odd integer (for example, a big prime or a product of two big primes). Then $\gcd(\mathfrak{R}, n) = \gcd(R, n) = 1$. Use the extended gcd algorithm to precompute $n' := -n^{-1} \pmod{\mathfrak{R}}$.

We associate $x \in \mathbb{Z}_n$ with $\bar{x} \in \mathbb{Z}_n$, where $\bar{x} \equiv xR \pmod{n}$. Since R is invertible modulo n, this association gives a bijection of $\mathbb{Z}_n$ onto itself. This bijection respects the addition in $\mathbb{Z}_n$: that is, $\overline{x+y} = \bar{x} + \bar{y}$ in $\mathbb{Z}_n$. Multiplication in $\mathbb{Z}_n$, on the other hand, corresponds to $\overline{xy} \equiv \bar{x}\bar{y}R^{-1} \pmod{n}$, and can be implemented as Algorithm 3.11 suggests.

Algorithm 3.11 **Montgomery multiplication**

Input: $\bar{x} = (x_{s-1} \dots x_0)_{\mathfrak{R}}$ and $\bar{y} = (y_{s-1} \dots y_0)_{\mathfrak{R}}$ (Montgomery representations of $x, y \in \mathbb{Z}_n$).
Output: Montgomery representation $\bar{z} = \bar{x}\bar{y}R^{-1} \pmod{n}$ of $z = xy \in \mathbb{Z}_n$.
Steps:

```
w̄ = (w_{2s-1} w_{2s-2} ... w_0)_ℜ := x̄ȳ.                    /* Integer product */
for (i = 0, ..., s − 1) {                                    /* Add a multiple of n to w̄ to force w_i = 0 */
    t := w_i n' (mod ℜ).                                     /* Single-precision operation */
    w̄ := w̄ + t n ℜ^i.                                        /* Multiple-precision operation */
}
ū := w̄/R.                                                    /* Shift by s words */
if (ū ⩾ n)  z̄ := ū − n, else z̄ := ū.
```

Montgomery multiplication works as follows. In the first step, it computes the integer product $\bar{w} := \bar{x}\bar{y}$. The subsequent for loop computes $\bar{w}R^{-1} \pmod{n}$. Since $n' \equiv -n^{-1} \pmod{\mathfrak{R}}$, the i-th iteration of the loop makes $w_i = 0$ (and leaves $w_{i-1}, \dots, w_0$ unchanged). So when the for loop terminates, we have $w_0 = w_1 = \cdots = w_{s-1} = 0$: that is, $\bar{w}$ is a multiple of $\mathfrak{R}^s = R$. Therefore, $\bar{u} = \bar{w}/R$ is an integer. Furthermore, this $\bar{w}$ is obtained by adding to $\bar{x}\bar{y}$ a multiple of n: that is, $\bar{u} = (\bar{x}\bar{y} + kn)/R$ for some integer $k \geqslant 0$. Since R is coprime to n, it follows that $\bar{u} \equiv \bar{x}\bar{y}R^{-1} \pmod{n}$. But this $\bar{u}$ may be bigger than the canonical representative of $\bar{z}$. Since k is an integer with s $\mathfrak{R}$-ary digits (so that $k < R$) and $\bar{x}\bar{y} < n^2$ and $n < R$, it follows that $\bar{u} < (n^2 + Rn)/R < 2n$. Therefore, if $\bar{u}$ exceeds $n-1$, a single subtraction suffices.

Computation of $\bar{x}\bar{y}$ requires $\leqslant s^2$ single-precision multiplications. One can use the optimized Algorithm 3.4 for that purpose. In case of squaring, $\bar{x} = \bar{y}$ and further optimizations (say, in the form of Karatsuba's method) can be employed.

Each iteration of the for loop carries out $s+1$ single-precision multiplications. (The reduction modulo $\mathfrak{R}$ is just returning the more significant word in the two-word product $w_i n'$.) Since, the for loop is executed s times, Algorithm 3.11 performs a total of $\leqslant s^2 + s(s+1) = 2s^2 + s$ single-precision multiplications.

Integer multiplication (Algorithm 3.4) followed by classical modular reduction (Algorithm 3.6) does almost an equal number of single-precision multiplications, but also $O(s)$ divisions of double-precision integers by single-precision ones. It turns out that the complicated for loop of Algorithm 3.6 is slower than the much simpler loop in Algorithm 3.11. But if precomputations in the Montgomery multiplication are taken into account, we do not tend to achieve a speed-up with this new technique. For modular exponentiations, however, precomputations need to be done only once: that is, outside the square-and-multiply loop, and Montgomery multiplication pays off. In Algorithm 3.12, we rewrite Algorithm 3.9 in terms of the Montgomery arithmetic. A similar rewriting applies to Algorithm 3.10.

Algorithm 3.12 **Montgomery exponentiation**

Input: $a \in \mathbb{Z}_n$, $e = (e_{r-1} \dots e_1 e_0)_2 \in \mathbb{N}$.
Output: $b = a^e \pmod n$.
Steps:
/* Precomputations */
$n' := -n \pmod{\mathfrak{R}}$. $\bar{a} := aR \pmod n$, $\bar{b} := R \pmod n$.

/* The square-and-multiply loop */
for $(i = r-1, \dots, 0)$ {
  $\bar{b} := \bar{b}^2 R^{-1} \pmod n$. /* Montgomery squaring */
  if $(e_i = 1)$ $\bar{b} := \bar{b}\bar{a}R^{-1} \pmod n$. /* Montgomery multiplication */
}
$b := \bar{b} \cdot 1 \cdot R^{-1} \pmod n$. /* Montgomery multiplication */

EXERCISE SET 3.3

3.8 Let $\mathfrak{R} \in \mathbb{N}$, $\mathfrak{R} > 1$. Show that every $a \in \mathbb{N}$ can be represented *uniquely* as a tuple $(a_{s-1}, \dots, a_1, a_0)$ for some $s \in \mathbb{N}$ (depending on a) with

$$a = a_{s-1}\mathfrak{R}^{s-1} + \cdots + a_1\mathfrak{R} + a_0,$$

$0 \leqslant a_i < \mathfrak{R}$ for all i and $a_{s-1} \neq 0$. In this case, we write a as $(a_{s-1} \dots a_0)_{\mathfrak{R}}$ or simply as $a_{s-1} \dots a_0$, when $\mathfrak{R}$ is understood from the context. $\mathfrak{R}$ is called the *radix* or *base* of this representation, $a_{s-1}, \dots, a_0$ the ($\mathfrak{R}$-ary) *digits* of a, a_{s-1} the *most significant digit*, a_0 the *least significant digit* and s the *size* of a with respect to the radix $\mathfrak{R}$.

3.9 Let $R \in \mathbb{N}$. Show that every $a \in \mathbb{N}$ can be written *uniquely* as

$$a = a_s R^s + a_{s-1}R^{s-1} + \cdots + a_1 R + a_0$$

with each $a_i \in \{1, 2, \dots, R\}$.

3.10 **Negative radix** Show that every integer $a \in \mathbb{Z}$ can be written as

$$a = a_s(-2)^s + a_{s-1}(-2)^{s-1} + \cdots + a_1(-2) + a_0$$

with $a_i \in \{0, 1\}$. Moreover, if we force that $a_s \neq 0$ for $a \neq 0$ and that $s = 0$ for $a = 0$, argue that this representation is unique.

3.11 Investigate the relative merits and demerits of the following three representations (in C) of multiple-precision integers needed for cryptography. In each case, we have room for storing 256 $\mathfrak{R}$-ary words, the actual size and a sign indicator. In the second and third representations, we use two extra locations (sizeIdx and signIdx) in the digit array for holding the size and sign information.

```
/* Representation 1 */
typedef struct {
   int size;
   boolean sign;
   ulong digits[256];
} cryptInt1;
```

```
/* Representation 2 */
typedef ulong cryptInt2[258];
#define signIdx 0
#define sizeIdx 1
```

```
/* Representation 3 */
typedef ulong cryptInt3[258];
#define signIdx 256
#define sizeIdx 257
```

Remark: We recommend the third representation.

3.12 Write an algorithm that prints a multiple-precision integer in decimal and an algorithm that accepts a string of decimal digits (optionally preceded by a $+$ or $-$ sign) and stores the corresponding integer as a multiple-precision integer. Also write algorithms for input and output of multiple-precision integers in hexadecimal, octal and binary.

3.13 Write an algorithm which, given two multiple-precision integers a and b, compares the absolute values $|a|$ and $|b|$. Also write an algorithm to compare a and b as signed integers.

3.14 **(a)** Write an algorithm that uses the Euclidean gcd loop (Proposition 2.15) to compute the gcd d of two integers a and b. (Observe that $\gcd(a, b) = \gcd(b, a \operatorname{rem} b)$ for $b \neq 0$.) **(b)** Modify the Euclidean gcd algorithm of Part (a), so that for given integers a, b we obtain d, u, v with $d = \gcd(a, b) = ua + vb$.

3.15 Describe a representation of rational numbers with exact multiple-precision numerators and denominators. Implement the arithmetic (addition, subtraction, multiplication and division) of rational numbers under this representation.

3.16 **Sliding window exponentiation** Suppose we want to compute the modular exponentiation $a^e \pmod n$. Consider the following variant of the square-and-multiply algorithm: Choose a small t (say, $t = 4$) and precompute $a^{2^{t-1}}, a^{2^{t-1}+1}, \ldots, a^{2^t-1}$ modulo n. Do squaring for every bit of e, but skip the multiplication for zero bits in e. Whenever a 1 bit is found, consider the next t bits of e (including the 1 bit). Let these t bits represent the integer l, $2^{t-1} \leqslant l \leqslant 2^t - 1$. Multiply by $a^l \pmod n$ (after computing usual t squares) and move right in e by t bit positions. Argue that this method works and write an algorithm based on this strategy. What are the advantages and disadvantages of this method over Algorithm 3.10?

3.17 Suppose we want to compute $a^e b^f \pmod n$, where both e and f are positive r-bit integers. One possibility is to compute a^e and b^f modulo n individually, followed by a modular multiplication. This strategy requires the running time of two exponentiations

(neglecting the time for the final multiplication). In this exercise, we investigate a trick to reduce this running time to something close to 1.25 times the time for one exponentiation. Precompute $ab \pmod n$. Inside the square-and-multiply loop, either skip the multiplication or multiply by a, b or ab, depending upon the next bits in the two exponents e and f. Complete the details of this algorithm. Deduce that, on an average, the running time of this algorithm is as declared above.

3.18 Let $m \in \mathbb{N}$, $m \neq 1$. An *addition chain* for m of length l is a sequence $1 = a_1, a_2, \ldots, a_l = m$ of natural numbers such that for every index i, $2 \leqslant i \leqslant l$, there exist indices $i_1, i_2 < i$ with $a_i = a_{i_1} + a_{i_2}$. (It is allowed to have $i_1 = i_2$.)
(a) If $1 = a_1, a_2, \ldots, a_l = m$ is an addition chain for m and if $j_1, j_2, \ldots, j_l$ is a permutation of $1, 2, \ldots, l$ with $a_{j_1} \leqslant a_{j_2} \leqslant \cdots \leqslant a_{j_l}$, show that $a_{j_1}, a_{j_2}, \ldots, a_{j_l}$ is also an addition chain for m. It, therefore, suffices to consider sorted addition chains only.
(b) Show that m has an addition chain of length $\leqslant 2 \lceil \lg m \rceil$. **[H]**
(c) Let G be a (multiplicative) group and $g \in G$. Design an algorithm for computing g^m given an addition chain for m. What is the complexity of the algorithm (in terms of the length of the given addition chain)?
(d) Show that Algorithms 3.9 and 3.10 use addition chains for e of lengths $\leqslant 2 \lceil \lg e \rceil$.

3.4 Elementary Number-theoretic Computations

Now that we know how to work in $\mathbb{Z}$ and in the residue class rings $\mathbb{Z}_n$, $n \in \mathbb{N}$, we address some important computational problems associated with these rings. In this chapter, we restrict ourselves only to those problems that are needed for setting up various cryptographic protocols.

3.4.1 Primality Testing

One of the simplest and oldest questions in algorithmic number theory is to decide if a given integer $n \in \mathbb{N}$, $n > 1$, is prime or composite. Practical primality testing algorithms are based on randomization techniques. In this section, we describe the Monte Carlo algorithm due to Miller and Rabin. The obvious question that comes next is to find one (or all) of the prime factors of an integer, deterministically or probabilistically proven to be composite. This is the celebrated *integer factorization problem* and will be formally introduced in Section 4.2. In spite of the apparent proximity between the primality testing and the integer factoring problems, they currently have widely different (known) complexities. Primality testing is easy and thereby promotes efficient setting up of cryptographic protocols. On the other hand, the difficulty of factoring integers protects these protocols against cryptanalytic attacks.

Definition 3.2 Let n be an odd integer greater than 1 and let $a \in \mathbb{Z}$ with $\gcd(a, n) = 1$. Then n is called a *pseudoprime to the base* a, if $a^{n-1} \equiv 1 \pmod n$. □

By Fermat's little theorem, a prime p is a pseudoprime to every base $a \in \mathbb{Z}$ with $\gcd(a, p) = 1$. However, the converse of this is not true. By Exercise 3.19, n is

not a pseudoprime to at least half of the bases in $\mathbb{Z}_n^*$, provided that there is at least one such base in $\mathbb{Z}_n^*$. Unfortunately, there exist composite integers m, known as *Carmichael numbers*, such that m is a pseudoprime to every base $a \in \mathbb{Z}_m^*$. The smallest Carmichael number is $561 = 3 \times 11 \times 17$. Exercises 3.21 and 3.22 investigate some properties of these numbers. Though Carmichael numbers are not very abundant in nature ($\mathbb{N}$), they are still infinite in number. So a robust primality test requires n to satisfy certain constraints in addition to being a pseudoprime to one or more bases. The following constraint is due to Solovay and Strassen.

Definition 3.3 Let n be an odd integer > 1 and let $a \in \mathbb{Z}$ with $\gcd(a, n) = 1$. Then n is called an *Euler pseudoprime* or a *Solovay–Strassen pseudoprime to the base* a, if $a^{(n-1)/2} \equiv \left(\frac{a}{n}\right) \pmod{n}$, where $\left(\frac{a}{n}\right)$ is the Jacobi symbol (Definition 2.32). Clearly, an Euler pseudoprime to the base a is also a pseudoprime to the base a. □

By Euler's criterion (Proposition 2.21), if p is a prime and $\gcd(a, p) = 1$, then p is an Euler pseudoprime to the base a. The converse in not true, in general, but if n is composite, then n is an Euler pseudoprime to at most $\phi(n)/2$ bases in $\mathbb{Z}_n^*$ (Exercise 3.20). This, in turn, implies that if n is an Euler pseudoprime to t randomly chosen bases in $\mathbb{Z}_n^*$, then the chance that n is composite is no more than $1/2^t$. This observation leads to a Monte Carlo algorithm for testing the primality of an integer, where the probability of error $(1/2^t)$ can be made arbitrarily small by choosing large values of t. A more efficient algorithm can be developed using the following concept due to Miller and Rabin.

Definition 3.4 Let n be an odd integer > 1 with $n - 1 = 2^r n'$, $r := \mathrm{v}_2(n-1) > 0$, n' odd, and let $a \in \mathbb{Z}$ with $\gcd(a, n) = 1$. Then n is called a *strong pseudoprime to the base* a, if either $a^{n'} \equiv 1 \pmod{n}$ or $a^{2^i n'} \equiv -1 \pmod{n}$ for some i, $0 \leqslant i < r$. It is clear that if n is a strong pseudoprime to the base a, then n is also a pseudoprime to the base a. What is less evident but still true is that if n is a strong pseudoprime to the base a, then n is also an Euler pseudoprime to the base a. □

The rationale behind this definition is the following. If for some $a \in \mathbb{Z}_n^*$ we have $a^{n-1} \not\equiv 1 \pmod{n}$, we conclude with certainty that n is composite. So assume that $a^{n-1} \equiv 1 \pmod{n}$ and consider the powers $b_i := a^{2^i n'} \pmod{n}$ for $i = 0, 1, \ldots, r$ to see how the sequence $b_0, b_1, \ldots$ eventually reaches $b_r \equiv 1 \pmod{n}$. If $b_0 \equiv 1 \pmod{n}$ already, this dynamics is clear. If, on the other hand, we have an i such that $b_i \not\equiv 1 \pmod{n}$, whereas $b_{i+1} \equiv 1 \pmod{n}$, then b_i is a square root of 1 modulo n. If n is a prime, the only square roots of 1 modulo n are ± 1 and so n must be a strong pseudoprime to the base a. On the other hand, if n is composite but not the power of a prime, then 1 has at least two non-trivial square roots (that is, square roots other than ± 1) modulo n (Exercise 3.30). We hope to find one such non-trivial square root of 1 in the sequence $b_0, b_1, \ldots, b_{r-1}$ and if we are successful, the compositeness of n is proved with certainty.

A complete residue system modulo an odd composite n contains at most $n/4$ bases to which n is a strong pseudoprime. The proof of this fact is somewhat involved (though elementary) and can be found elsewhere, for example, in Chapter V of Koblitz [153]. Here, we concentrate on the Monte Carlo Algorithm 3.13 known as the *Miller–Rabin primality test* and based on this observation.

Algorithm 3.13 **Miller–Rabin primality test**

Input: An odd integer $n \in \mathbb{N}$ and an acceptable probability δ of failure.
Output: A certificate that either "n is composite" or "n is prime".
Steps:
Find out n' and r such that $n - 1 = 2^r n'$ with $r \in \mathbb{N}$ and n' odd.
Determine the number t of iterations, so that the probability of failure is $\leqslant \delta$.
for ($j = 1, \ldots, t$) {
  Choose a random base a, $1 < a < n$.
  $b := a^{n'} \pmod n$. /* Compute b_0 */
  if ($b \not\equiv 1 \pmod n$) {
    $i := 0$.
    while ($i < r - 1$) and ($b \not\equiv -1 \pmod n$) {
      $i{+}{+}$, $b := b^2 \pmod n$. /* Compute b_i by squaring b_{i-1} */
      if ($b \equiv 1 \pmod n$) { Return "n is composite". }
    }
    if ($b \not\equiv -1 \pmod n$) { Return "n is composite". }
  }
}
Return "n is prime".

Whenever Algorithm 3.13 outputs n *is composite*, it is correct. On the other hand, if it certifies n as *prime*, there is a probability δ that n is composite. This probability can be made very small by choosing a suitably large value of the iteration count t. For cryptographic applications, $\delta \leqslant 1/2^{80}$ is considered sufficiently safe. In view of the first statement of the last paragraph, we can take $t = 40$ to meet this error bound. In practice, much smaller values of t offer the desired confidence. For example, if n is of bit length 250, 500, 750 or 1000, the respective values $t = 12$, 6, 4 and 3 suffice.

Although, in Algorithm 3.13, we have chosen a to be an arbitrary integer between 2 and $n - 2$, there is apparently no harm, if we choose a randomly in the interval $2 \leqslant a < 2^{32}$. In fact, such a choice of single-precision bases is desirable, because that makes the exponentiation $a^{n'} \pmod n$ more efficient (See Algorithm 3.9). A typical cryptographic application loads at start-up a precalculated table of small primes (say, the first thousand primes). Choosing the bases randomly from this list of small primes is indeed a good idea.

Deterministic primality proving

While the Miller–Rabin algorithm settles the primality testing problem in a practical sense, it is, after all, a randomized algorithm. It is interesting, at the minimum theoretically, to investigate the deterministic complexity of primality testing. There has been a good amount of research in this line. Let us sketch here the history of deterministic primality proving, without going to rigorous mathematical details.

One natural strategy to check for the primality of a positive integer n is to factor it. However, factoring integers is a computationally difficult problem. Primality proving

has been found to be a much easier computational exercise. That is, one need not factorize n explicitly in order to claim about the primality of n.

The (seemingly) first modern primality testing algorithm is due to Miller[204]. This algorithm is deterministic polynomial-time, provided that the extended Riemann hypothesis or ERH (Conjecture 2.3) is true. Since the ERH is still an unsolved problem in mathematics, it cannot be claimed with certainty if Miller's test is really a polynomial-time algorithm. Rabin [248] provided a version of Miller's test which is unconditionally polynomial-time, but is, at the same time, randomized. This is what we have discussed earlier under the name Miller–Rabin primality test. This is a Monte Carlo algorithm which produces the answer *no* (composite) with certainty, but the answer *yes* (prime) with some (small) probability of error. Solovay and Strassen's test [287] based on Definition 3.3 is another *no*-biased randomized polynomial-time primality test and can be made deterministically polynomial-time under the ERH.

Adleman and Huang [3], using the work of Goldwasser and Kilian [116], provide a *yes*-biased randomized primality-proving algorithm that runs in *expected* polynomial time unconditionally. Adleman et al. [4] propose the first deterministic algorithm that runs unconditionally in time less than fully exponential (in $\log n$). Its (worst) running time is $(\ln n)^{\mathrm{O}(\ln\ln\ln n)}$, which is still not polynomial. (The exponent $\ln\ln\ln n$ grows very slowly with n, but still is not a constant.)

In August 2002, Agarwal, Kayal and Saxena came up with the first deterministic primality testing algorithm that runs in polynomial time unconditionally, that is, under no unproven assumptions. This algorithm, popularly abbreviated as the *AKS algorithm*, is based on the observation that n is prime if and only if $(X+a)^n \equiv X^n + a \pmod n$ for every $a \in \mathbb{Z}$ (Exercise 3.26). A naive application of this observation requires computing an exponential number of coefficients in the binomial expansion of $(X+a)^n$. The AKS algorithm gets around with this difficulty by checking the new congruence

$$(X+a)^n \equiv X^n + a \pmod{n, h(X)} \tag{3.2}$$

for some polynomial $h(X)$ of small degree. Here the notation $(\mathrm{mod}\ n, h(X))$ means modulo the ideal $\langle n, h(X)\rangle = n\mathbb{Z}[X] + h(X)\mathbb{Z}[X]$ of $\mathbb{Z}[X]$. If $\deg h(X)$ is bounded by a polynomial in $\log n$, $(X+a)^n$ (and also X^n+a) can be computed modulo $n, h(X)$ in polynomial time. However, reduction modulo $h(X)$ may allow a composite n to satisfy the new congruence. Agarwal et al. took $h(X) := X^r - 1$ for some prime $r = \mathrm{O}(\ln^6 n)$ with $r-1$ having a prime divisor $\geqslant 4\sqrt{r}\ln n$. From a result in analytic number theory due to Fouvry, such a prime r always exists. Congruence (3.2) is verified for this $h(X)$ and for at most $2\sqrt{r}\ln n$ values of a. An elementary proof presented in Agarwal et al. [5] demonstrates that this suffices to conclude deterministically and unconditionally about the primality of n. The AKS algorithm in this form runs in time $\tilde{\mathrm{O}}(\ln^{12} n)$.

Lenstra and Pomerance [175] have reduced the running time of the AKS algorithm to $\tilde{\mathrm{O}}(\ln^6 n)$. The AKS paper comes with another conjecture which, if true, yields a $\tilde{\mathrm{O}}(\ln^3 n)$ deterministic primality-proving algorithm.

Conjecture 3.1 **AKS conjecture** Let n be an odd integer > 1, and $r \in \mathbb{P}$ with $r \nmid n$. If

$$(X-1)^n \equiv X^n - 1 \pmod{n, X^r - 1},$$

then either n is prime or $n^2 \equiv 1 \pmod r$. □

It remains an open question whether a future version of the AKS algorithm would supersede the Miller–Rabin test in terms of performance. As long as the answers are not favourable to the AKS algorithm, these new theoretical endeavours do not seem to have sufficient impacts on cryptography. Primes certified by the Miller–Rabin test are at present secure enough for all applications. Nonetheless, the AKS breakthrough has solid theoretical implications and deserves mention in a prime context.

3.4.2 Generating Random Primes

If a random prime of a given bit length t is called for, we can keep on generating random odd integers of bit length t and check these integers for primality using the Miller–Rabin test. The prime number Theorem 2.20 ascertains that after $\mathrm{O}(t)$ iterations we expect to find a prime. A somewhat similar but reasonably faster algorithm is discussed in Exercise 4.14. We will henceforth call random primes of a given bit length and having no additional imposed properties as *naive primes*. Naive primes are often not cryptographically secure, because the primes used in many protocols should satisfy certain properties in order to preclude some known cryptanalytic attacks.

Definition 3.5 Let p be an odd prime. Then p is called a *safe prime*, if $(p-1)/2$ is also a prime, whereas p is called a *strong prime*, if

(1) $p-1$ has a large prime divisor, say, q,
(2) $p+1$ has a large prime divisor, say, q', and
(3) $q-1$ has a large prime divisor, say, q''.

In cryptography, a *large* prime divisor typically refers to one with bit length $\geqslant 160$. □

A random safe prime of a given bit length t can be found by generating a random sequence of natural numbers n congruent to 3 modulo 4 and of bit length t, until one is found for which both n and $(n-1)/2$ are primes (as certified by the Miller–Rabin primality test). The prime number theorem once again implies that this search is expected to terminate after $\mathrm{O}(t^2)$ iterations.

For generating a random strong prime p of bit length t, we first generate q' and q'' and then q and finally p. (See the notations of Definition 3.5.) Algorithm 3.14 describes Gordon's algorithm in which the bit lengths l and l' of q and q' are nearly $t/2$ and the bit length l'' of q'' is slightly smaller than l'. In our concrete implementation of the algorithm, we choose $l := \lceil t/2 \rceil - 2$, $l' := \lfloor t/2 \rfloor - 20$ and $l'' := \lceil t/2 \rceil - 22$. If t is sufficiently large (say, $t \geqslant 400$), the prime divisors q, q' and q'' are then cryptographically large.

The simple check that Gordon's algorithm correctly computes a strong prime of bit length t with q, q' and q'' as in Definition 3.5 is based on Fermat's little theorem and is left to the reader. Note that with our choice of l, l' and l'', the loop variables i and j run through single-precision values only, thereby making arithmetic involving them efficient. Also note that the ranges over which i and j vary are sufficiently large so that we *expect* the (outer) while loop to be executed only once. This implementation has a tendency to generate *smaller* values of q and p (with the given bit sizes). In practice, this is not a serious problem and can be avoided, if desired, by choosing random values of i and j from the indicated ranges.

Algorithm 3.14 **Gordon's strong-prime generator**

Input: $t \in \mathbb{N}$, $t \geqslant 400$.
Output: A strong prime p of bit length t.
Steps:
$l := \lceil t/2 \rceil - 2$, $l' := \lfloor t/2 \rfloor - 20$, $l'' := \lceil t/2 \rceil - 22$.
while (1) {
  Find a (random) naive prime q' of bit length l'.
  Find a (random) naive prime q'' of bit length l''.
  for ($i = \lceil (2^{l-1} - 1)/2q'' \rceil, \ldots, \lfloor (2^l - 2)/2q'' \rfloor$) { /* Search for q */
    $q := 2iq'' + 1$.
    if (q is prime) {
      $p' := 2((q')^{q-2} \bmod q)q' - 1$.
      for ($j = \lceil (2^{t-1} - p')/2qq' \rceil, \ldots, \lfloor (2^t - 1 - p')/2qq' \rfloor$) { /* Search for p */
        $p := p' + 2jqq'$.
        if (p is prime) { Return }
      }
    }
  }
}

Gordon's algorithm takes only nominally more expected running time than that needed by the algorithm discussed at the beginning of Section 3.4.2 for generating naive primes of the same bit length. On the other hand, safe primes are much costlier to generate and may be avoided, unless the situation specifically demands their usage.

3.4.3 Modular Square Roots

Determination of square roots modulo a prime p is frequently needed in cryptographic applications. In this section, we assume that p is an odd prime and want to compute the square roots of $a \in \mathbb{Z}$, $\gcd(a, p) = 1$, modulo p, provided that a is a quadratic residue modulo p, that is, if $\left(\frac{a}{p}\right) = 1$. Using the Jacobi symbol the value $\left(\frac{a}{p}\right)$ can be computed efficiently as Algorithm 3.15 suggests.

The correctness of Algorithm 3.15 follows from the properties of the Jacobi symbol (Proposition 2.22 and Theorem 2.19). The value of $(-1)^{(b^2-1)/8}$ is determined by the value of b modulo 8, that is, by the three least significant bits of b:

$$(-1)^{(b^2-1)/8} = \begin{cases} +1 & \text{if } b \equiv \pm 1 \pmod 8, \\ -1 & \text{if } b \equiv \pm 3 \pmod 8. \end{cases}$$

Similarly, $(-1)^{(a-1)(b-1)/4}$ can be computed using only the second least significant bits of a and b as:

$$(-1)^{(a-1)(b-1)/4} = \begin{cases} -1 & \text{if } a \equiv b \equiv 3 \pmod 4, \\ +1 & \text{otherwise.} \end{cases}$$

If $\left(\frac{a}{p}\right) = 1$, our next task is to compute $x \in \mathbb{Z}_p$ with $x^2 \equiv a \pmod p$. If one such x is found, the other square root of a modulo p is $-x \equiv p - x \pmod p$. If

Algorithm 3.15 **Computation of the Legendre symbol**

Input: An odd prime p and an integer a, $1 \leqslant a < p$.

Output: The Legendre symbol $\left(\frac{a}{p}\right)$.

Steps:

$b := p$, $k := 1$. /* Initialize */

/* The Euclidean loop */
while $(a \neq 1)$ {
  $v := \mathrm{v}_2(a)$, $a := a/2^v$. /* Strip out powers of 2 from a */
  if (v is odd) $k := (-1)^{(b^2-1)/8}k$. /* Multiply by $\left(\frac{2}{b}\right)$, if necessary */
  $k := (-1)^{(a-1)(b-1)/4}k$. /* Use the reciprocity law */
  $(a, b) := (b \operatorname{rem} a, a)$. /* $\left(\frac{b}{a}\right) = \left(\frac{b \operatorname{rem} a}{a}\right)$ */
}
Return k.

$p \equiv 3 \pmod 4$ or $p \equiv 5 \pmod 8$, we have explicit formulas for a square root x. The remaining case, namely $p \equiv 1 \pmod 8$, is somewhat complicated. In this case, we use the probabilistic algorithm due to Tonelli and Shanks. The details are given in Algorithm 3.16. The explicit formulas for the first two cases are easy to verify. We now prove the correctness of the algorithm in the remaining case.

Since $\mathbb{Z}_p^*$ is cyclic and has order $p - 1 = 2^v q$, the 2-Sylow subgroup G of $\mathbb{Z}_p^*$ has order 2^v and is also cyclic. Let g be a generator of G. By Euler's criterion, a^q is a square in G and, therefore, $a^q g^e = 1$ (in G) for some even integer e, $0 \leqslant e < 2^v$, and $x \equiv a^{(q+1)/2} g^{e/2} \pmod p$ is a square root of a modulo p.

A generator g of G can be obtained by choosing random elements b from $\mathbb{Z}_p^*$ and computing the Legendre symbol $\left(\frac{b}{p}\right)$. It is easy to see that $b^q \in G$. Furthermore, b^q is a generator of G if and only if $\left(\frac{b}{q}\right) = -1$. Finding a quadratic non-residue in $\mathbb{Z}_p^*$ is the probabilistic part of the algorithm. Since exactly half of the elements of $\mathbb{Z}_p^*$ are quadratic non-residues, one expects to find one after a few random trials. In order to make the exponentiation b^q efficient, b should be chosen as single-precision integers. The while loop of the algorithm computes the multiplier $g^{e/2}$ in x using $\mathrm{O}(v)$ iterations by successively locating the 1 bits of e starting from the least significant end.

To sum up, square roots modulo a prime can be computed in probabilistic polynomial time. Computing square roots modulo a composite integer n is, on the other hand, a very difficult problem, unless the complete factorization of n is known (see Section 4.2 and Exercise 3.29).

EXERCISE SET 3.4

3.19 Let $n \in \mathbb{N}$ be odd and composite and suppose that there exists (at least) one $a \in \mathbb{Z}_n^*$ with $a^{n-1} \not\equiv 1 \pmod n$. Show that $b^{n-1} \not\equiv 1 \pmod n$ for at least half of the bases $b \in \mathbb{Z}_n^*$. **[H]**

Algorithm 3.16 **Modular square root**

Input: An odd prime p and an integer a, $1 \leqslant a < p$.
Output: A square root of a modulo p (if existent).
Steps:
if $(\left(\frac{a}{p}\right) = -1)$ { Return "a does not have a square root modulo p". }

if $(p \equiv 3 \pmod 4)$ { Return $a^{\frac{p+1}{4}} \pmod p$. }

if $(p \equiv 5 \pmod 8)$
  if $(a^{\frac{p-1}{4}} \equiv 1 \pmod p)$ { Return $a^{\frac{p+3}{8}} \pmod p$ }
  else { Return $2a(4a)^{\frac{p-5}{8}} \pmod p$. }

/* The case $p \equiv 1 \pmod 8$ */
$v := \mathrm{v}_2(p-1)$, $q := (p-1)/2^v$. /* q is odd */
Find a random quadratic non-residue b modulo p and set $g := b^q \pmod p$.
$x := a^{(q+1)/2} \pmod p$.
Precompute $a^{-1} \pmod p$.
while (1) {
  find the smallest $i \in \{0, 1, \ldots, v-1\}$ for which $(x^2 a^{-1})^{2^i} \equiv 1 \pmod p$.
  if $(i = 0)$ { Return x. }
  $x := xg^{2^{v-i-1}} \pmod p$.
}

3.20 Let $n \in \mathbb{N}$ be odd and composite.
(a) Show that there exists $a \in \mathbb{Z}_n^*$, such that $a^{(n-1)/2} \not\equiv \left(\frac{a}{n}\right) \pmod n$. **[H]**
(b) Show that $b^{(n-1)/2} \not\equiv \left(\frac{b}{n}\right) \pmod n$ for at least half of the bases $b \in \mathbb{Z}_n^*$. **[H]**

3.21 Let $n \in \mathbb{N}$ be a Carmichael number, that is, a composite integer for which $a^{n-1} \equiv 1 \pmod n$ for all a coprime to n, that is, $\mathrm{ord}_n(a) | (n-1)$ for all $a \in \mathbb{Z}_n^*$. Prove that:
(a) $(p-1)|(n-1)$ for every prime divisor p of n. **[H]**
(b) n is odd. **[H]**
(c) n is square-free. **[H]**
(d) n has at least three distinct prime divisors.

3.22 **(a)** Let $n \in \mathbb{N}$ be a square-free composite integer, such that $(p-1)|(n-1)$ for every prime divisor p of n. Show that n is a Carmichael number.
(b) Demonstrate that $561 = 3 \times 11 \times 17$; $2{,}821 = 7 \times 13 \times 31$; and $172{,}081 = 7 \times 13 \times 31 \times 61$ are Carmichael numbers.
(c) Assume that for some $k \in \mathbb{N}$ the integers $p_1 := 6k+1$, $p_2 := 12k+1$ and $p_3 := 18k+1$ are prime. Prove that $p_1 p_2 p_3$ is a Carmichael number.
(d) Deduce that $1{,}729 = 7 \times 13 \times 19$ and $294{,}409 = 37 \times 73 \times 109$ are Carmichael numbers.

3.23 **Fermat's test for prime numbers** Let $n \in \mathbb{N}$ and let $n - 1 = p_1^{e_1} \cdots p_r^{e_r}$, $e_i \in \mathbb{N}$, be the prime factorization of $n-1$. Suppose that there exist integers $a_1, \ldots, a_r$ such

that for each i we have $a_i^{n-1} \equiv 1 \pmod{n}$ and $a_i^{(n-1)/p_i} \not\equiv 1 \pmod{n}$. Show that n is prime.

3.24 Pépin's test for Fermat numbers Show that the Fermat number $n := 2^{2^k} + 1$ is prime if and only if $3^{(n-1)/2} \equiv -1 \pmod{n}$.

3.25 Write an algorithm that, given natural numbers t, l with $l < t$, outputs a (probable) prime p of bit length t such that $p - 1$ has a (probable) prime divisor q of bit length l.

3.26 Let $n \in \mathbb{N}$.
(a) Show that the ring $\mathbb{Z}[X]/n\mathbb{Z}[X]$ is (canonically) isomorphic to the ring $\mathbb{Z}_n[X]$. In view of this, we write $f(X) \equiv g(X) \pmod{n}$ to mean either that the coefficients of f are congruent modulo n to the respective coefficients of g or that the polynomials $f(X)$ and $g(X)$ are congruent modulo the principal ideal of $\mathbb{Z}[X]$ generated by n.
(b) Prove that if n is a prime, then $(X + a)^n \equiv X^n + a \pmod{n}$ for every $a \in \mathbb{Z}$.
(c) Prove that for composite n there exists $k \in \mathbb{N}$, $1 < k < n$, with $n \nmid \binom{n}{k}$. Deduce that in this case $(X + a)^n \not\equiv X^n + a \pmod{n}$ for some $a \in \mathbb{Z}$.
(d) Let $h(X) \in \mathbb{Z}[X]$ and let $\bar{h}(X) \in \mathbb{Z}_n[X]$ be the canonical image of $h(X)$ in $\mathbb{Z}_n[X]$. Show that the ring $\mathbb{Z}[X]/\langle n, h(X)\rangle$ is isomorphic to the ring $\mathbb{Z}_n[X]/\langle \bar{h}(X)\rangle$.

3.27 Modify Algorithm 3.15 to compute the (generalized) Jacobi symbol $\left(\frac{a}{b}\right)$ for odd $b \in \mathbb{N}$ and for arbitrary $a \in \mathbb{Z}$.

3.28 Implement the Chinese remainder theorem for integers, that is, write an algorithm that takes as input pairwise relatively prime moduli $n_1, \ldots, n_r \in \mathbb{N}$ and integers $a_i \in \mathbb{Z}_{n_i}$ for $i = 1, \ldots, r$ and that outputs $a \in \mathbb{Z}_{n_1 \cdots n_r}$ with $a \equiv a_i \pmod{n_i}$ for all $i = 1, \ldots, r$. **[H]**

3.29 Let $f(X)$ be a non-constant polynomial in $\mathbb{Z}[X]$.
(a) Let the congruence $f(x) \equiv 0 \pmod{p^e}$, $e \in \mathbb{N}$, have a solution $x \equiv a \pmod{p^e}$. Show that if an integer $a' := a + kp^e$ solves the congruence $f(x) \equiv 0 \pmod{p^{e+1}}$, then k satisfies the congruence

$$f'(a)k \equiv -f(a)/p^e \pmod{p}.$$

Here $f(a)/p^e$ means integer division. Demonstrate that this congruence may have 0, 1 or p solutions (for k) depending on the values of $f'(a)$ and $f(a)/p^e$. Each such k gives a solution a' of $f(x) \equiv 0 \pmod{p^{e+1}}$ with $a' \equiv a \pmod{p^e}$. We say that the solution a' (modulo p^{e+1}) is obtained from the solution a (modulo p^e) by (*Hensel*) *lifting*.
(b) Lifting together with the Chinese remainder theorem allow us to reduce the problem of solving a polynomial congruence modulo an arbitrary modulus $n \in \mathbb{N}$ to the problem of solving the same congruence modulo the prime divisors of n. More precisely, if the prime factorization $n = p_1^{\alpha_1} \cdots p_r^{\alpha_r}$ of n and all the solutions of the congruences $f(x) \equiv 0 \pmod{p_i}$ for all $i = 1, \ldots, r$ are given, design an algorithm to compute all the solutions of the congruence $f(x) \equiv 0 \pmod{n}$.

3.30 Let $n \in \mathbb{N}$ be odd and $a \in \mathbb{Z}$. Deduce that the congruence $x^2 \equiv a \pmod{n}$ has exactly $\prod_{\substack{p \in \mathbb{P} \\ p \mid n}} \left[1 + \left(\frac{a}{p}\right)\right]$ solutions modulo n.

3.31 Show that Algorithm 3.17 correctly computes $\lfloor\sqrt{n}\rfloor$ for $n \in \mathbb{N}$. Specify a strategy to initialize a before the while loop. Determine how Algorithm 3.17 can be used to check if a given $n \in \mathbb{N}$ is a perfect square. **[H]**

Algorithm 3.17 **Integer square root**

Input: $n \in \mathbb{N}$.
Output: $\lfloor\sqrt{n}\rfloor$.
Steps:
Using bit operations initialize a to an integral value x, $\lfloor\sqrt{n}\rfloor \leqslant x \leqslant 2\lfloor\sqrt{n}\rfloor$.
while (1) { /* Newton's iteration loop */
  $b := \lfloor(a + \lfloor n/a \rfloor)/2\rfloor$.
  if $(a \leqslant b)$ { Return a. }
  $a := b$.
}

3.32 **(a)** Design an algorithm that, given $n, k \in \mathbb{N}$, computes $\lfloor\sqrt[k]{n}\rfloor = \lfloor n^{1/k}\rfloor$. **[H]**
(b) Design an algorithm to check if a given $n \in \mathbb{N}$ is an integral power of another integer.

3.5 Arithmetic in Finite Fields

Many cryptographic protocols are based on the (apparent) intractability of the discrete logarithm problem (Section 4.2) in the multiplicative group of a finite field $\mathbb{F}_q$. The arithmetic of the finite fields $\mathbb{F}_p$, $p \in \mathbb{P}$, and $\mathbb{F}_{2^n}$, $n \in \mathbb{N}$, is easy to implement and run efficiently. In view of this, these two kinds of finite fields are most popular in cryptography and we concentrate our algorithmic study on these fields only.

A prime field $\mathbb{F}_p$ is the quotient ring $\mathbb{Z}_p = \mathbb{Z}/\langle p\rangle$. In Section 3.3.4, we have already made a thorough study of the arithmetic of the rings $\mathbb{Z}_n$, $n \in \mathbb{N}$. We recall that the elements of $\mathbb{F}_p$ are represented as integers from the set $\{0, 1, \ldots, p-1\}$ and the arithmetic in $\mathbb{F}_p$ is the modulo p integer arithmetic. Since p is typically multiple-precision, the characteristic p of $\mathbb{F}_p$ is odd. The fields of even characteristic that we will study are the non-prime fields $\mathbb{F}_{2^n}$.

Section 2.9.3 explains several representations of extension fields. The most common one is the polynomial-basis representation $\mathbb{F}_{2^n} \cong \mathbb{F}_2[X]/\langle f(X)\rangle$ for an irreducible polynomial $f(X)$ of degree n in $\mathbb{F}_2[X]$. In that case, an element of $\mathbb{F}_{2^n}$ has the canonical representation as a polynomial $a_0 + a_1X + \cdots + a_{n-1}X^{n-1}$, $a_i \in \mathbb{F}_2$, of degree $< n$. An arithmetic operation on two elements of $\mathbb{F}_{2^n}$ is the same operation in $\mathbb{F}_2[X]$ followed by reduction modulo the *defining polynomial* $f(X)$. So we start with the implementation of the polynomial arithmetic over $\mathbb{F}_2$.

3.5.1 Arithmetic in the Ring $\mathbb{F}_2[X]$

A polynomial over $\mathbb{F}_2$ (or any field) is identified by its coefficients of which only finitely many are non-zero. Thus for storing a polynomial $g(X) = a_dX^d + a_{d-1}X^{d-1} + \cdots +$

$a_1X + a_0$ it is sufficient to store the finite ordered sequence $a_d a_{d-1} \ldots a_1 a_0$. It is not necessary to demand $a_d \neq 0$, but the shortest sequence representing a non-zero polynomial corresponds to $a_d \neq 0$ and in this case $\deg g = d$. On the other hand, as we see later it is often useful to pad such a sequence with *leading* zero coefficients. As an example, the polynomial $X^2 + 1 \in \mathbb{F}_2[X]$ is representable as 101 or as 0101 or as 00101 or $\cdots$.

Since $\mathbb{F}_2$ can be viewed as the set $\{0, 1\}$ with operations modulo 2, a polynomial in $\mathbb{F}_2[X]$ is essentially a bit string unique up to insertion (and deletion) of leading zero bits. As in the case of multiple-precision integers, we pack these coefficients in an array of 32-bit words and maintain the number of coefficients *belonging to* the polynomial. For example, the polynomial $g(X) = X^{64} + X^{31} + X^7 + 1$ can be stored in an array $w_2 w_1 w_0$ of three 32-bit words. w_0 consists of the coefficients of $X^0, X^1, \ldots, X^{31}$, w_1 consists of the coefficients of $X^{32}, X^{33}, \ldots, X^{63}$, and w_2 consists of the coefficient of X^{64}. It is up to the implementation scheme to decide whether the coefficients are to be stored from left to right or from right to left in the bits of a word. We assume that less significant coefficients go to the less significant bits of a word. For the polynomial g above, the word w_0 viewed as an unsigned integer will then be $w_0 = 2^{31} + 2^7 + 1$, whereas we have $w_1 = 0$. The least significant bit of w_2 would be 1. The remaining 31 bits of w_2 are not important and can be assigned any value as long as we maintain the information that only the coefficients of X^i, $0 \leqslant i \leqslant 64$, need to be considered. On the other hand, if we want to store the coefficients of g upto that of X^{80}, then the bits of w_2 at locations $1, \ldots, 16$ must be zero, whereas those at locations $17, \ldots, 31$ may be of any value. We, however, always recommend the use of leading zero-bits to fill the portion of the leading word not belonging to the polynomial.

Such a representation of elements of $\mathbb{F}_2[X]$, in addition to being compact, facilitates efficient implementation of arithmetic functions. As we will shortly see, we need not often extract the individual coefficients of a polynomial but apply bit operations on entire words to process 32 coefficients simultaneously per operation. We usually do not need polynomials of degrees > 4096 for cryptographic applications. It is, therefore, sufficient to declare a *static* array capable of storing all the 8193 coefficients of a product of two such *largest* polynomials. The zero polynomial may be represented as one with zero word size, whereas the degree of the zero polynomial is taken to be $-\infty$ which may be representable as -1.

We now describe the arithmetic functions on two *non-zero* polynomials

$$a(X) = a_r X^r + \cdots + a_1 X + a_0 \text{ and } b(X) = b_s X^s + \cdots + b_1 X + b_0. \quad \textbf{(3.3)}$$

Under our implementation, a and b demand $\rho := \lceil (r+1)/32 \rceil$ and $\sigma := \lceil (s+1)/32 \rceil$ machine words $\alpha_{\rho-1} \ldots \alpha_1 \alpha_0$ and $\beta_{\sigma-1} \ldots \beta_1 \beta_0$. We also assume paddings with leading zero bits in the areas not belonging to the operands.

Note that the addition of $\mathbb{F}_2$ is the same as the XOR ($\oplus$) of two bits. Applying this bit operation on words α_i and β_i adds 32 coefficients of the operand polynomials simultaneously (see Algorithm 3.18). Finally note that $-1 = 1$ in any field of characteristic 2, that is, subtraction is the same as addition in such a field.

The product $a(X)b(X)$ can be computed as in Algorithm 3.19. Once again, using wordwise operations yields faster implementation. By AND and OR, we denote the bit-wise *and* and *or* operations on 32-bit words. The easy verification of the correctness

of this algorithm is left to the reader. As in the case of addition, one might want to make the polynomial c compact after its words $\gamma_{\tau-1}, \ldots, \gamma_0$ are computed.

Algorithm 3.18 **Polynomial addition**

Input: $a(X), b(X) \in \mathbb{F}_2[X]$ as in Equation (3.3).
Output: $c(X) = a(X) + b(X)$ (to be stored in the array $\gamma_{\tau-1} \ldots \gamma_1\gamma_0$).
Steps:
$\tau := \max(\rho, \sigma)$.
for $(i = 0, \ldots, \min(\rho, \sigma) - 1)$ $\gamma_i := \alpha_i \oplus \beta_i$.
if $(\rho > \sigma)$ for $(i = \sigma, \ldots, \rho - 1)$ $\gamma_i := \alpha_i$,
else if $(\rho < \sigma)$ for $(i = \rho, \ldots, \sigma - 1)$ $\gamma_i := \beta_i$.
while $(\tau > 0)$ and $(\gamma_{\tau-1} = 0)$ $\tau--$. /* Make c compact (optional) */

Algorithm 3.19 **Polynomial multiplication**

Input: $a(X), b(X) \in \mathbb{F}_2[X]$ as in Equation (3.3).
Output: $c(X) = a(X)b(X)$ (to be stored in the array $\gamma_{\tau-1} \ldots \gamma_1\gamma_0$).
Steps:
$\tau := \rho + \sigma - 1$. /* The size of the product */
for $(i = 0, \ldots, \tau - 1)$ $\gamma_i := 0$. /* Initialize the product */

```
/* The quadratic multiplication loop */
for (k = 0, ..., 31) {    /* For each bit position in a word */
  for (j = 0, ..., σ − 1) {    /* For each word of b */
    if (b_j AND 2^k) {    /* if the k-th bit of b_j is 1 */
      for (i = 0, ..., ρ − 1) {    /* For each word of a */
        set γ_{i+j} := γ_{i+j} ⊕ (a_i ≪ k) and γ_{i+j+1} := γ_{i+j+1} ⊕ (a_i ≫ (32 − k)).
      }
    }
  }
}
```

The square of $a(X) \in \mathbb{F}_2[X]$ can be computed very easily using the fact that

$$a(X)^2 = (a_r X^r + \cdots + a_1 X + a_0)^2 = a_r X^{2r} + \cdots + a_1 X^2 + a_0.$$

This gives us a linear-time (in terms of r or ρ) algorithm instead of the quadratic general-purpose multiplication Algorithm 3.19. We leave the implementational details to the reader.

Division with remainder in $\mathbb{F}_2[X]$ is implemented in Algorithm 3.20. As before, we continue to work with the operands $a(X)$ and $b(X)$ as in Equation (3.3). But now we make a further assumption that $b_s = 1$, so that $\beta_{\sigma-1} \neq 0$, and also that $s \leqslant r$. When the Euclidean division loop of Algorithm 3.20 terminates, the array locations $\delta_{\sigma-1}, \ldots, \delta_1, \delta_0$ contain the remainder. The arrays γ and δ may be made compact to discard the leading zero bits, if any.

Algorithm 3.20 **Euclidean division of polynomials**

Input: $a(X), b(X) \in \mathbb{F}_2[X]$ as in Equation (3.3).
Output: $c(X) = a(X) \text{ quot } b(X)$ (to be stored in the array $\gamma_{\tau-1} \ldots \gamma_1\gamma_0$) and $d(X) = a(X) \text{ rem } b(X)$ (to be sored in the array $\delta_{\rho-1} \ldots \delta_1\delta_0$).
Steps:

```
τ := ⌈(s − r + 1)/32⌉.    /* The size of the quotient */
for i = 0, ..., τ − 1 { γ_i := 0 }    /* Initialize c(X) to 0 */
for i = 0, ..., ρ − 1 { δ_i := α_i }    /* Copy a(X) to d(X) */
/* Euclidean division loop */
for i = r, ..., s {
  if (the coefficient of X^i in d(X) is 1) {
    j := (i − s) quot 32, k := (i − s) rem 32.
    /* Set the coefficient of X^{i−s} of c(X) */
    γ_j := γ_j OR 2^k.
    /* Update d(X) := d(X) − X^{i−s} b(X) */
    for l = 0, ..., σ − 1 {
      δ_{l+j} := δ_{l+j} ⊕ (b_l ≪ k).
      δ_{l+j+1} := δ_{l+j+1} ⊕ (b_l ≫ (32 − k)).
    }
  }
}
```

Computing modular inverses requires computation of extended gcds of polynomials in $\mathbb{F}_2[X]$. We again start with the non-zero polynomials $a(X), b(X) \in \mathbb{F}_2(X)$ and compute polynomials $d(X)$, $u(X)$ and $v(X)$ in $\mathbb{F}_2[X]$ with $d(X) = \gcd(a(X), b(X)) = u(X)a(X) + v(X)b(X)$, $\deg u < \deg b$ and $\deg v < \deg a$. For polynomials, we do not have an equivalent of the binary gcd algorithm (Algorithm 3.8). We use repeated Euclidean divisions instead.

The proof for the correctness of Algorithm 3.21 is similar to that for Algorithm 3.8. Here, we introduce the variables r_k, U_k and V_k for $k = 0, 1, 2, \ldots$. The initialization goes as: $r_0 := a$, $r_1 := b$, $U_0 := 1$, $U_1 := 0$, $V_0 := 0$ and $V_1 := 1$. During the k-th iteration ($k = 1, 2, \ldots$), we first use Euclidean division to get $r_{k-1} = q_k r_k + r_{k+1}$ which gives $r_{k+1} = r_{k-1} - q_k r_k$. We also compute $U_{k+1} = U_{k-1} - q_k U_k$ and $V_{k+1} = V_{k-1} - q_k V_k$ using the values available from the previous two iterations so as to maintain the relation $r_{k+1} = U_{k+1} r_0 + V_{k+1} r_1$ for all $k = 1, 2, \ldots$. In Algorithm 3.21, the k-th iteration of the while loop begins with $x = r_{k-1}$, $y = r_k$, $u_1 = U_k$ and $u_2 = U_{k-1}$ and ends after updating the values to $x = r_k$, $y = r_{k+1}$, $u_1 = U_{k+1}$ and $u_2 = U_k$. It is not necessary to maintain the values V_k in the main loop. After the loop terminates, one computes $V_k = (r_k - U_k r_0)/r_1$.

Modular arithmetic in $\mathbb{F}_2[X]$ is very much similar to the modular arithmetic in $\mathbb{Z}$. If $f(X)$ is a non-constant polynomial of $\mathbb{F}_2[X]$ (not necessarily irreducible), we represent elements of $\mathbb{F}_2[X]/\langle f \rangle$ as polynomials in $\mathbb{F}_2[X]$ of degrees $< n$. Given two

Algorithm 3.21 **Extended gcd of polynomials**

Input: Nonzero polynomials $a, b \in \mathbb{F}_2[X]$.
Output: Polynomials $d, u, v \in \mathbb{F}_2[X]$ satisfying
$$d = \gcd(a, b) = ua + vb, \quad \deg u < \deg b, \quad \deg v < \deg a.$$
Steps:
/* Initialize */
$x := a$, $y := b$, $u_1 := 1$, $u_2 := 0$.

/* Repeated Euclidean division */
while $(y \neq 0)$ {
  Simultaneously compute $q := x \operatorname{quot} y$ and $r := x \operatorname{rem} y$ (Algorithm 3.20).
  $u := u_2 - qu_1$, $u_2 := u_1$, $u_1 := u$,
  $x := y$, $y := r$.
}
$d := x$, $v := (d - ua)/b$.

such polynomials a and b, we compute the sum $a + b$ simply as the sum in $\mathbb{F}_2[X]$. The product ab is computed by first computing the product ab in $\mathbb{F}_2[X]$ and then computing the remainder of Euclidean division of this product by f. Inverse of a modulo f exists if and only if $\gcd(a, f) = 1$ (in $\mathbb{F}_2[X]$). In that case, extended gcd computation gives us polynomials u, v such that $1 = ua + vf$, so that $ua \equiv 1 \pmod f$. If $a \neq 0$, then Algorithm 3.21 computes u with $\deg u < \deg f = n$, so that we take this u to be the canonical representative of a^{-1} in $\mathbb{F}_2[X]/\langle f \rangle$. Finally, for $e \in \mathbb{N}$ the computation of the modular exponentiation $a^e \pmod f$ can be done using an algorithm very similar to Algorithm 3.9 or Algorithm 3.10. We leave the details to the reader.

3.5.2 Finite Fields of Characteristic 2

For the polynomial basis representation $\mathbb{F}_{q^n} \cong \mathbb{F}_q[X]/\langle f \rangle$, we need an irreducible polynomial $f(X) \in \mathbb{F}_q[X]$ of degree n. We shortly present a probabilistic algorithm that generates a random monic irreducible polynomial in $\mathbb{F}_q[X]$ of given degree $n \in \mathbb{N}$. Although we are interested only in the case $q = 2$, this algorithm holds even if q is any arbitrary prime or an arbitrary prime power.

First, we describe a deterministic polynomial-time algorithm for checking the irreducibility of a non-constant polynomial $f \in \mathbb{F}_q[X]$ (over $\mathbb{F}_q$). If f is reducible, it has a factor of degree $i \leqslant \lfloor n/2 \rfloor$. Also recall (Theorem 2.40, p82) that $X^{q^i} - X$ is the product of all monic irreducible polynomials of $\mathbb{F}_q[X]$ of degrees dividing i. Therefore, if f has an irreducible factor of degree i, then $\gcd(f, X^{q^i} - X) = \gcd(f, X^{q^i} - X \operatorname{rem} f)$ will be a non-constant polynomial. Algorithm 3.22 employs these simple observations.

Now, recall from Section 2.9.2 that a random monic polynomial of $\mathbb{F}_q[X]$ of degree n is irreducible with probability approximately $1/n$. Therefore, if we keep on checking for irreducibility random monic polynomials in $\mathbb{F}_q[X]$ of degree n, then after $\mathrm{O}(n)$ checks we expect to find an irreducible polynomial. This leads to the Las Vegas probabilistic Algorithm 3.23.

Algorithm 3.22 **Check for irreducibility of a polynomial**

Input: A non-constant polynomial $f \in \mathbb{F}_q[X]$.
Output: A (deterministic) certificate whether f is irreducible or not.
Steps:
$n := \deg f$, $g := X$.
for $i = 1, \ldots, \lfloor n/2 \rfloor$ {
  $g := g^q \pmod f$. /* Here $g = X^{q^i} \text{ rem } f$ */
  if $(\deg(\gcd(f, g - X)) > 0)$ { Return "f is reducible". }
}
Return "f is irreducible".

Algorithm 3.23 **Generation of a random irreducible polynomial**

Input: $n \in \mathbb{N}$, $n \geqslant 2$.
Output: A random monic irreducible polynomial $f \in \mathbb{F}_q[X]$ of degree n.
Steps:
while (1) {
  $f :=$ a random monic polynomial in $\mathbb{F}_q[X]$ of degree n.
  if (f is irreducible) { Return }
}

Once the defining irreducible polynomial f is available, we carry out the arithmetic in $\mathbb{F}_{2^n} = \mathbb{F}_2[X]/\langle f \rangle$ as modular polynomial arithmetic with respect to the modulus f. This is described at the end of Section 3.5.1. Since this modular arithmetic involves taking the remainder of Euclidean division by f, it is sometimes expedient to choose f to be an irreducible polynomial of certain special types. The randomized algorithm described above gives a random monic irreducible polynomial f of degree n having on an average $\approx n/2$ non-zero coefficients. The division algorithm (Algorithm 3.20) in that case takes time $\mathrm{O}(n^2)$. On the other hand, if f is a sparse polynomial (like a trinomial), the Euclidean division loop can be rewritten to exploit this sparsity, thereby bringing down the running time of the division procedure to $\mathrm{O}(n)$. (See Exercise 3.34. Also see Exercise 3.38 for computing isomorphisms between different polynomial-basis representations of the same field.)

Let p be a prime and let $n \in \mathbb{N}$. We have seen how to implement arithmetic in $\mathbb{F}_p$ and hence by Exercise 3.35 that in $\mathbb{F}_p[X]$ too. If $f(X) \in \mathbb{F}_p[X]$ is an irreducible polynomial of degree n and if $q = p^n$, then $\mathbb{F}_q = \mathbb{F}_{p^n} = \mathbb{F}_p[X]/\langle f(X) \rangle$ and we implement the arithmetic of $\mathbb{F}_q$ as the polynomial arithmetic of $\mathbb{F}_p[X]$ modulo f. Again by Exercise 3.35, this gives us the arithmetic of $\mathbb{F}_q[Y]$. Now, for $m \in \mathbb{N}$ and a monic irreducible polynomial $g(Y) \in \mathbb{F}_q[Y]$ we have a representation $\mathbb{F}_{q^m} = \mathbb{F}_{p^{nm}} \cong \mathbb{F}_q[Y]/\langle g(Y) \rangle$. Instead of having such a two-way representation of $\mathbb{F}_{p^{nm}}$ we may also represent $\mathbb{F}_{p^{nm}}$ as $\mathbb{F}_p[Z]/\langle h(Z) \rangle$, where $h(Z) \in \mathbb{F}_p[Z]$ is a monic irreducible polynomial of degree nm. It usually turns out that the second representation of $\mathbb{F}_{p^{nm}}$

is more efficient. However, there are some situations where the two-way representation performs better. This is, in particular, the case when the arithmetic of $\mathbb{F}_q$ can be made more efficient than the modular polynomial arithmetic of $\mathbb{F}_p[X]/\langle f(X)\rangle$. For example, we might precompute tables of arithmetic operations of $\mathbb{F}_q$ and use table lookups for performing the coefficient arithmetic of $\mathbb{F}_q[Y]$. This demands $\mathrm{O}(q^2)$ storage and is feasible only when q is small. On the other hand, if we find a primitive element γ of $\mathbb{F}_q^*$ and precompute a table that maps $i \mapsto \gamma^i$ and another that maps $\gamma^i \to i$, then products in $\mathbb{F}_q$ can be computed in time $\mathrm{O}(1)$ using table lookups. If, in addition, we store the Zech's logarithm table (Section 2.9.3) for $\mathbb{F}_q$, then addition in $\mathbb{F}_q$ can also be performed in $\mathrm{O}(1)$ time with table lookup. Both these three tables take $\mathrm{O}(q)$ memory which (though better than $\mathrm{O}(q^2)$ storage of the previous scheme) is feasible only for small q.

3.5.3 Selecting Suitable Finite Fields

Not all finite fields are suitable for cryptographic applications. In this section, we discuss the desirable properties of a field $\mathbb{F}_q$ so that secured protocols on $\mathbb{F}_q$ can be developed. We first note that such protocols are usually based on the apparent intractability of the so-called *discrete logarithm problem* (*DLP*) (Section 4.2). As a result, selections of suitable fields are dictated by the known cryptanalytic algorithms to solve the DLP (See Section 4.4). We shall mostly concentrate on $\mathbb{F}_q$ with either $q = p$ a prime or $q = 2^n$ for some $n \in \mathbb{N}$. By the *bit size* of q, denoted $|q|$, we mean the number of bits in the binary representation of q, that is, $|q| = \lceil \lg q \rceil$. As we have seen, each element of $\mathbb{F}_q$ is representable using $O(|q|)$ bits and, therefore, $|q|$ is often also called the *size* of $\mathbb{F}_q$.

The first requirement on a cryptographically suitable field $\mathbb{F}_q$ is that the size $|q|$ should be sufficiently large. Recent cryptanalytic studies show that sizes $|q| \leqslant 512$ are not secure enough. Sizes $|q| \geqslant 768$ are recommended for secure applications. For long-term security, one might even require $|q| \geqslant 2048$.

Any field of the recommended size is, however, not adequately secure. The cardinality $\#F_q = q$ must be such that $q - 1$ has at least one large prime divisor q' (See the Pohlig–Hellman method in Section 4.4). By *large*, we usually mean $|q'| \geqslant 160$. In addition, this prime factor q' of $q - 1$ should be known to us. If $q = p$ is a prime, then a safe prime or a strong prime serves our purpose (Definition 3.5, Algorithm 3.14). Also see Exercise 3.25. On the other hand, if $q = 2^n$, the only way to obtain q' is by factorizing the Mersenne number $M_n := q - 1 = 2^n - 1$. Factorizing M_n for $n \geqslant 768$ is a very difficult task. Luckily, extensive tables of complete or partial factorizations of M_n are available. For example, for $n = 769$ (a prime number), we have

$$M_{769} = 2^{769} - 1 = 1{,}591{,}805{,}393 \times 6{,}123{,}566{,}623{,}856{,}435{,}977{,}170{,}641 \times q',$$

where q' is a 657-bit prime. These tables should be consulted for choosing a suitable value of n.

The multiplicative group $\mathbb{F}_q^*$ is cyclic (Theorem 2.38). If the complete integer factorization of $q - 1$ is known, then it is possible to find, in polynomial time (in $|q|$), a primitive element of $\mathbb{F}_q^*$. Algorithm 3.24 computes $r = \mathrm{O}(\lg n)$ exponentiations in G in order to conclude whether a given element $a \in G$ is a generator of G. For $G =$

$\mathbb{F}_q^*$, we have polynomial-time exponentiation algorithms, so Algorithm 3.24 runs in deterministic polynomial time. By Exercise 2.47, the probability of a randomly chosen element of G being primitive is $\phi(m)/m$. In view of the lower bound on $\phi(m)/m$, given in Theorem 3.1 and proved by Rosser and Schoenfield [253], Algorithm 3.25 is expected to return a random primitive element of G after $\mathrm{O}(\ln \ln m)$ iterations.

Theorem 3.1 Let $m \in \mathbb{N}$, $m \geqslant 5$. Then $\phi(m)/m \geqslant 1/(6 \ln \ln m)$. ■

Algorithm 3.24 **Check for primitive element**

Input: A cyclic group G of cardinality $\#G = m$ with known factorization $m = p_1^{\alpha_1} \cdots p_r^{\alpha_r}$ and an element $a \in G$.
Output: A deterministic certificate that a is a generator of G.
Steps:
/* We assume that G is multiplicatively written and has the identity e */
for $i = 1, \ldots, r$ {
  if $(a^{(n-1)/p_i} = e)$ { Return "a is not a generator of G". }
}
Return "a is a generator of G".

Algorithm 3.25 **Computation of a generator of a finite cyclic group**

Input: A cyclic group G of cardinality $\#G = m$ with known factorization $m = p_1^{\alpha_1} \cdots p_r^{\alpha_r}$.
Output: A generator g of G.
Steps:
while (1) {
  $g :=$ a random element of G.
  if (g is a generator of G) /* Algorithm 3.24 */ { Return }
}

If, however, the factorization of $\#G = m$ is not known, there are no known (deterministic or probabilistic) algorithms for finding a random generator of G or even for checking if a given element of G is primitive. This is indeed one of the intractable problems of computational algebraic number theory. This problem for $G = \mathbb{F}_q^*$ can be bypassed as follows.

Recall that we have chosen q in such a way that $\#\mathbb{F}_q^* = q - 1$ has a large known prime factor q'. Let H be the unique subgroup of G of order q'. Then H is also cyclic and we choose to work in H (using the arithmetic of G). It turns out that if $q' \geqslant 2^{160}$ and if H is not contained in a *proper* subfield of $\mathbb{F}_q$, the security of cryptographic protocols over $\mathbb{F}_q$ does not degrade too much by the use of H (instead of the full G) as the ground group. But we now face a new problem, that is, the problem of finding a generator of H. Since $\#H = q'$ is a prime, every element of $H \setminus \{1\}$ is a generator of H. So the problem essentially reduces to that of finding any non-identity element of H. This latter problem has a simple probabilistic solution. First of all, if $q - 1 = q'$ is itself

prime, choosing any random non-identity element of $\mathbb{F}_q^*$ will do. So assume $q' < q-1$. Choose a random $a \in \mathbb{F}_q^*$ and let $b := a^{(q-1)/q'}$. By Lagrange's theorem (Theorem 2.2, p24), $b^{q'} = a^{q-1} = 1$ and, therefore, by Proposition 2.5 $b \in H$. Now, $\mathbb{F}_q$ being a field, the polynomial $X^{(q-1)/q'} - 1 \in \mathbb{F}_q[X]$ can have at most $(q-1)/q'$ roots in $\mathbb{F}_q$ (that is, in $\mathbb{F}_q^*$) and hence the probability that $b = 1$ is $\leqslant ((q-1)/q')/(q-1) = 1/q'$. This justifies the randomized polynomial running time of the Las Vegas Algorithm 3.26. Indeed if $q' \geqslant 2^{160}$, the while loop of the algorithm is executed only once almost always.

Algorithm 3.26 **Computation of an element of given order**

Input: A finite field $\mathbb{F}_q$ and an (odd) prime factor q' of $q-1$ with $q' < q-1$.
Output: An element $b \in \mathbb{F}_q^*$ of multiplicative order q'.
Steps:

```
while (1) {
   a := a random element of F_q \ {0, ±1}.
   b := a^{(q-1)/q'}.
   if (b ≠ 1) { Return }
}
```

3.5.4 Factoring Polynomials over Finite Fields

Polynomial factorization over finite fields is an interesting computational problem. All deterministic algorithms known for this purpose are quite poor: that is, fully exponential in the size of the field. However, if randomization is allowed, we have reasonably efficient (polynomial-time) algorithms. In this section, we outline the basic working of the modern probabilistic algorithms for polynomial factorization over finite fields. We assume that a non-constant polynomial $f \in \mathbb{F}_q[X]$ is to be factored. Without loss of generality, we can take f to be *monic*. We assume further that the arithmetic of $\mathbb{F}_q$ and that of $\mathbb{F}_q[X]$ is available. We work with a general value of $q = p^n$, p prime and $n \in \mathbb{N}$, though in some cases we have to treat the case $p = 2$ separately. Irreducibility (or otherwise) in this section means the same over $\mathbb{F}_q$.

The factorization algorithm we are going to discuss is a generalization of the root finding algorithm (see Exercise 3.36) and consists of three steps:

Square-free factorization (SFF) Decompose the input polynomial f into a product of square-free polynomials.

Distinct-degree factorization (DDF) Given a square-free polynomial f of degree d, compute $f = f_1 \cdots f_d$ with each f_i being a product of irreducible polynomials of degree i.

Equal-degree factorization (EDF) Given a product f of irreducible polynomials of the same degree, find out the irreducible factors of f.

We now provide a separate detailed discussion for each of these three steps.

Square-free factorization

Theorem 3.2 is at the very heart of the square-free factorization algorithm and is a generalization of Exercise 2.61.

Theorem 3.2 Let K be a field and $f \in K[X]$ a non-constant monic polynomial. Then the polynomial $f/\gcd(f, f')$ is square-free, where f' is the formal derivative of f. In particular, f is square-free if and only if $\gcd(f, f') = 1$.

Proof Let $f = f_1^{\alpha_1} \cdots f_r^{\alpha_r}$ be the factorization of f with pairwise distinct monic irreducible polynomials $f_1, \ldots, f_r$, $r \in \mathbb{N}$, with $\alpha_1, \ldots, \alpha_r \in \mathbb{N}$ and with $0 \neq a \in K$. In order to determine $\mathrm{v}_{f_1}(f')$, we employ the usual rules for derivatives to get $f' = f_1^{\alpha_1 - 1} \cdots f_r^{\alpha_r - 1}(\alpha_1 f_1' f_2 \cdots f_r + f_1 g)$ for some $g \in K[X]$. If $\alpha_1 = 0 \in K$, then $\mathrm{v}_{f_1}(f') \geqslant \alpha_1$. Otherwise, $\mathrm{v}_{f_1}(f') = \alpha_1 - 1$, since $\gcd(f_1, f_1') = 1 = \gcd(f_1, f_i)$, $i > 1$. Similar is the case for $\mathrm{v}_{f_i}(f')$ for $i = 2, \ldots, r$. It follows that $\gcd(f, f') = f_1^{\beta_1} \cdots f_r^{\beta_r}$, where each $\beta_i \in \{\alpha_i - 1, \alpha_i\}$, so that $f/\gcd(f, f') = f_1^{\delta_1} \cdots f_r^{\delta_r}$, $\delta_i \in \{0, 1\}$, is square-free. ■

The algorithm for SFF over $\mathbb{F}_q$ is now almost immediate except for one subtlety, namely, the consideration of the case $f/\gcd(f, f') = 1$, or equivalently, $f' = 0$. In order to see when this case can occur, let us write the non-zero terms of f as $f = a_1 X^{e_1} + \cdots + a_t X^{e_t}$ with distinct exponents $e_1, \ldots, e_t$ and $a_i \in \mathbb{F}_q^*$. Then $f' = a_1 e_1 X^{e_1 - 1} + \cdots + a_t e_t X^{e_t - 1} = 0$ if and only if $e_1 = e_2 = \cdots = e_t = 0 \in \mathbb{F}_q$, that is, if p divides all of $e_1, \ldots, e_t$. But then $f(X) = h(X)^p$, where $h(X) = a_1^{p^{n-1}} X^{e_1/p} + \cdots + a_r^{p^{n-1}} X^{e_r/p}$, since $a_i^q = a_i^{p^n} = a_i$ for all i. These observations motivate the recursive Algorithm 3.27. It is easy to check that this (deterministic) algorithm runs in time polynomially bounded by $\deg f$ and $\log q$.

Algorithm 3.27 **Square-free factorization**

Input: A monic non-constant polynomial $f \in \mathbb{F}_q[X]$, $q = p^n$, p prime, $n \in \mathbb{N}$.
Output: A square-free factorization of f.
Steps:
Compute f'.
if $(f' = 0)$ {
  Compute $h \in \mathbb{F}_q[X]$ such that $f = h^p$.
  Recursively compute a SFF $h = h_1 \cdots h_s$ of h.
  Return the SFF of f as $f = (h_1 \cdots h_s)(h_1 \cdots h_s) \cdots (h_1 \cdots h_s)$ (p times).
} else {
  Recursively compute a SFF $\gcd(f, f') = g_1 \cdots g_s$ of $\gcd(f, f')$.
  Return the SFF of f as $f = (f/\gcd(f, f'))g_1 \cdots g_s$.
}

Distinct-degree factorization

Let $f \in \mathbb{F}_q[X]$ be a square-free polynomial of degree d. We can write $f = f_1 \cdots f_d$, where for each i the polynomial $f_i \in \mathbb{F}_q[X]$ is the product of all the irreducible factors of f of degree i. If f does not have an irreducible factor of degree i, then we take $f_i = 1$ as usual.[5] In order to compute the polynomials f_i, we make use of the fact that

[5] Conventionally, an empty product is taken to be the multiplicative identity and an empty sum to be the additive identity.

$\mathcal{I}_{q,i}(X) = X^{q^i} - X$ is the product of all monic irreducible polynomials in $\mathbb{F}_q[X]$ whose degrees divide i (see Theorem 2.40 on p 82). It immediately follows that $\gcd(f_i, \mathcal{I}_{q,i}) = \prod_{j|i} f_j$. Thus a few (at most d) gcd computations give us all f_i. But the polynomials $\mathcal{I}_{q,i}$ are of rather large degrees. But since $\gcd(\mathcal{I}_{q,i}, f) = \gcd(\mathcal{I}_{q,i} \bmod f, f)$, keeping polynomials reduced modulo f implies that we take gcds of polynomials of degrees $\leqslant d$. This, in turn, implies that the DDF can be performed in (deterministic) polynomial time (in d and $\ln q$).

Algorithm 3.28 shows an implementation of the DDF. Though the algorithm does not require f to be monic, there is no harm in assuming so.

Algorithm 3.28 **Distinct-degree factorization**

Input: A (non-constant) square-free polynomial $f \in \mathbb{F}_q[X]$.
Output: The DDF of f, that is, the polynomials $f_1, \ldots, f_d$ as explained above.
Steps:

```
g := f.   /* Make a local copy of f */
h = X. i = 1.
while (deg g ≠ 0) {
   h := h^q (mod f).    /* Modular exponentiation */
   f_i := gcd(h − X, g).
   g := g/f_i.    /* Factor out f_i from g */
   i++.
}
if (i < d) { f_{i+1} := 1, ..., f_d := 1. }
```

This simple-minded implementation of the DDF is theoretically not the most efficient one known. In fact, it turns out that the DDF (and not the seemingly more complicated EDF) is the bottleneck of the entire polynomial factorization process. Therefore, making the DDF more efficient is important and there are lots of improvements suggested in the literature. All these improved algorithms essentially do the same thing as above (that is, the computation of $\gcd(\mathcal{I}_{q,i}, f)$), but they optimize the computation of the polynomials $\mathcal{I}_{q,i} \bmod f$. The best-known method (due to Kaltofen and Shoup) is based on the observation that, in general, most of the f_i are 1. Therefore, instead of computing each $\gcd(\mathcal{I}_{q,i}, f)$, one may break the interval $1, \ldots, d$ into several subintervals $I_1, I_2, \ldots, I_l$ and compute $F_j := \gcd(\prod_{i \in I_j} \mathcal{I}_{q,i}, f)$, $j = 1, \ldots, l$. Only those F_j that turn up to be non-constant are further decomposed.

For cryptographic purposes, we will, however, deal with rather small values of $d = \deg f$. (Typically d is at most a few thousands.) The asymptotically better algorithms usually do not outperform the simple Algorithm 3.28 for these values of d.

Equal-degree factorization

Equal-degree factorization, the last step of the polynomial factorization process, is the only probabilistic part of the algorithm. We may assume that f is a (monic) square-free polynomial of degree d and that each irreducible factor of f has the same (known) degree, say δ. If $d = \delta$, then f is irreducible. So we assume that $d > \delta$, that is, $d = r\delta$ for some $r \in \mathbb{N} \setminus \{1\}$. Theorem 3.3 provides the basic foundations for the EDF.

Theorem 3.3 Let g be any polynomial in $\mathbb{F}_q[X]$ and let $\delta \in \mathbb{N}$. Then $X^{q^\delta} - X$ divides $g^{q^\delta} - g$.

Proof If $g = 0$, there is nothing to prove. If $g = a_l X^l + \cdots + a_1 X + a_0 \neq 0$ with $a_i \in \mathbb{F}_q$, then $g^{q^\delta} - g = a_l(X^{lq^\delta} - X^l) + \cdots + a_1(X^{q^\delta} - X)$. It is easy to verify that $X^{q^\delta} - X$ divides $X^{iq^\delta} - X^i$ for every $i \in \mathbb{N}$. ■

Now, we have to separate two cases, namely, q is odd and q is even. Theorem 3.3 is valid for any q, even or odd, but taking q as odd allows us to write $g^{q^\delta} - g = g(g^{(q^\delta-1)/2}-1)(g^{(q^\delta-1)/2}+1)$. With the above assumptions on f we have $f|(X^{q^\delta}-X)$ and, therefore, $f|(g^{q^\delta} - g)$, so that $f = \gcd(g^{q^\delta} - g, f) = \gcd(g, f)\gcd(g^{(q^\delta-1)/2} - 1, f)\gcd(g^{(q^\delta-1)/2}+1, f)$. If g is randomly chosen, then $\gcd(g^{(q^\delta-1)/2} - 1, f)$ is with probability $\approx 1/2$ a non-trivial factor of f. The idea is, therefore, to keep on choosing random g and computing $\bar{f} := \gcd(g^{(q^\delta-1)/2} - 1, f)$ until one gets $0 < \deg \bar{f} < \deg f$. One then recursively applies the algorithm to $\bar{f}$ and $f/\bar{f}$. It is sufficient to choose g with $\deg g < 2\delta$. Obviously, the exponentiation g^{q^δ} has to be carried out modulo f. We leave the details to the reader, but note that trying $\mathrm{O}(1)$ random polynomials g is expected to split f and, therefore, the EDF runs in expected polynomial time.

For the case $q = 2^n$, essentially the same algorithm works, but we have to use the split $g^{q^\delta} + g = g^{2^{n\delta}} + g = (g^{2^{n\delta-1}} + g^{2^{n\delta-2}} + \cdots + g^2 + g)(g^{2^{n\delta-1}} + g^{2^{n\delta-2}} + \cdots + g^2 + g + 1)$. Once again computing $\gcd(g^{2^{n\delta-1}} + g^{2^{n\delta-2}} + \cdots + g^2 + g, f)$ for a random $g \in \mathbb{F}_q[X]$ splits f with probability $\approx 1/2$ and, thus, we get an EDF algorithm that runs in expected polynomial time.

EXERCISE SET 3.5

3.33 Find a (polynomial-basis) representation of $\mathbb{F}_{27}$. Compute a primitive element in this representation.

3.34 **(a)** Show that the running time of Algorithm 3.20 is $\mathrm{O}(s(r-s))$ which reaches the maximum order of $\mathrm{O}(r^2) = \mathrm{O}(s^2)$, when $s \approx r/2$.
(b) Suppose b is known to have e non-zero coefficients. Modify the Euclidean division loop of Algorithm 3.20 so that the algorithm runs in time $\mathrm{O}((r-s)e)$. **[H]** In particular, if $e = \mathrm{O}(1)$, the running time of Algorithm 3.20 becomes linear, namely $\mathrm{O}(r)$.

3.35 Implement the polynomial arithmetic of $\mathbb{F}_q[X]$ given that of $\mathbb{F}_q$.

3.36 Let $q = p^n$ (p prime and $n \in \mathbb{N}$), $f(X) \in \mathbb{F}_q[X]$ a non-constant polynomial and let $g := \gcd(f, X^q - X)$.
(a) If S is the set of all roots of f in $\mathbb{F}_q$, show that $g = \prod_{a \in S}(X - a)$. Thus, g is a square-free polynomial which splits over $\mathbb{F}_q$ and has the same roots (over $\mathbb{F}_q$) as f.
If $\deg g = 0$ or 1, then we know all the roots of g and hence of f. So, for the rest of this exercise, we assume that $\deg g \geqslant 2$.
(b) Consider the case that p is *odd*. Let $b \in \mathbb{F}_q$ be arbitrary. Show that

$$(X + b)((X + b)^{(q-1)/2} - 1)((X + b)^{(q-1)/2} + 1) = X^q - X$$

and that

$$g = \gcd(g, X+b)\gcd(g, (X+b)^{(q-1)/2} - 1)\gcd(g, (X+b)^{(q-1)/2} + 1).$$

Explain how Algorithm 3.29 produces two non-trivial factors of g (over $\mathbb{F}_q$) in probabilistic polynomial time. **[H]** Write an algorithm to compute all the roots of f in $\mathbb{F}_q$.

Algorithm 3.29 **Computing roots of a polynomial: odd characteristic**

Input: A square-free polynomial $g \in \mathbb{F}_q[X]$ that splits over $\mathbb{F}_q$.
Output: Polynomials $g_1, g_2 \in \mathbb{F}_q[X]$ with $g = g_1 g_2$ and $\deg g_i \geqslant 1$ for $i = 1, 2$.
Steps:
if $(g(0) = 0)$ { $(g_1, g_2) := (X, g(X)/X)$, return. }
while (1) {
  Select a random element $b \in \mathbb{F}_q$.
  $h := (X+b)^{(q-1)/2} - 1 \pmod{g}$.
  $g_1 := \gcd(g, h)$.
  if $(1 \leqslant \deg g_1 < \deg g)$ { $g_2 := g/g_1$, return. }
}

(c) Now, assume that $p = 2$ and define the polynomial

$$H(X) := X + X^2 + X^4 + \cdots + X^{2^{n-1}} \in \mathbb{F}_q[X].$$

Let $b \in \mathbb{F}_q$ be arbitrary. Show that

$$H(X+b)(H(X+b)+1) = X^q - X$$

[H] and that

$$g(X) = \gcd(g(X), H(X+b))\gcd(g(X), H(X+b)+1).$$

Explain how Algorithm 3.30 produces two non-trivial factors of g (over $\mathbb{F}_q$) in probabilistic polynomial time. Write an algorithm to compute all the roots of f in $\mathbb{F}_q$.

Algorithm 3.30 **Computing roots of a polynomial: characteristic 2**

Input: A square-free polynomial $g \in \mathbb{F}_q[X]$ that splits over $\mathbb{F}_q$.
Output: Polynomials $g_1, g_2 \in \mathbb{F}_q[X]$ with $g = g_1 g_2$ and $\deg g_i \geqslant 1$ for $i = 1, 2$.
Steps:
if $(g(0) = 0)$ { $(g_1, g_2) := (X, g(X)/X)$, return. }
while (1) {
  Select a random element $b \in \mathbb{F}_q$.
  $h := (X+b) + (X+b)^2 + (X+b)^4 + \ldots + (X+b)^{2^{n-1}} \pmod{g}$.
  $g_1 := \gcd(g, h)$.
  if $(1 \leqslant \deg g_1 < \deg g)$ { $g_2 := g/g_1$, return. }
}

3.37 Use Exercise 3.36 to compute all the roots of the following polynomials:

(a) $X^6 + 6X^4 + 4X^2 + 6$ in $\mathbb{F}_7[X]$.
(b) $X^3 + (\alpha^2 + \alpha)X^2 + (\alpha^2 + \alpha + 1)$ in $\mathbb{F}_8[X]$, where $\mathbb{F}_8$ is represented as $\mathbb{F}_2(\alpha)$, α being a root of the polynomial $X^3 + X + 1$.

3.38 Let f and g be two monic irreducible polynomials over $\mathbb{F}_q$ and of the same degree $n \in \mathbb{N}$. Consider the two representations $\mathbb{F}_{q^n} \cong \mathbb{F}_q[X]/\langle f(X)\rangle \cong \mathbb{F}_q[Y]/\langle g(Y)\rangle$. In this exercise, we study how we can compute an isomorphism between these two representations. The polynomial $f(Y)$ splits into linear factors over $\mathbb{F}_q[Y]/\langle g(Y)\rangle$. Consider a root $\alpha = \alpha(Y)$ of $f(Y)$ in $\mathbb{F}_q[Y]/\langle g(Y)\rangle$. Show that $1, \alpha, \alpha^2, \ldots, \alpha^{n-1}$ is an $\mathbb{F}_q$-basis of (the $\mathbb{F}_q$-vector space) $\mathbb{F}_q[Y]/\langle g(Y)\rangle$. For $i = 0, \ldots, n-1$, write (uniquely) $\alpha^i = \sum_{j=0}^{n-1} \alpha_{ij} Y^j$ with $\alpha_{ij} \in \mathbb{F}_q$, and consider the matrix $\mathrm{A} = (\alpha_{ij})_{0 \leqslant i \leqslant n-1, 0 \leqslant j \leqslant n-1}$. Show that the map $\mathbb{F}_q[X]/\langle f(X)\rangle \to \mathbb{F}_q[Y]/\langle g(Y)\rangle$ that maps (the equivalence class of) $a_0 + a_1 X + \cdots + a_{n-1}X^{n-1}$ to (the equivalence class of) $b_0 + b_1 Y + \cdots + b_{n-1}Y^{n-1}$, where $(b_0 b_1 \ldots b_{n-1}) = (a_0 a_1 \ldots a_{n-1})\mathrm{A}$, is an $\mathbb{F}_q$-isomorphism.

3.39 Let $q = p^n$ for a prime p and $n \in \mathbb{N}$. We have seen that the elements of $\mathbb{F}_p$ can be represented as integers between 0 and $p-1$, whereas the elements of $\mathbb{F}_q = \mathbb{F}_p[X]/\langle f\rangle$ can be represented as polynomials modulo some irreducible polynomial $f \in \mathbb{F}_p[X]$ of degree n, that is, as polynomials of $\mathbb{F}_p[X]$ of degrees $< n$. Show that the substitution $X = p$ in the polynomial representation of elements of $\mathbb{F}_q$ gives a representation of elements of $\mathbb{F}_q$ as integers between 0 and $q-1$. We call this latter representation of elements of $\mathbb{F}_q$ the *packed representation*. Compare the advantages and disadvantages of the packed representation over the polynomial representation.

3.40 Let G be a cyclic multiplicatively written group of order m (and with the identity element e). Assume that the factorization of $m = p_1^{\alpha_1} \cdots p_r^{\alpha_r}$ is known. Devise an algorithm that computes the order of an arbitrary element in G. **[H]**

3.41 Berlekamp's Q-matrix factorization Let $f(X) \in \mathbb{F}_q[X]$ be a monic square-free polynomial of degree d, that admits a factorization $f(X) = f_1(X) \ldots f_r(X)$ with each $f_i(X) \in \mathbb{F}_q[X]$ monic, non-constant and irreducible. (Note that f_i are pairwise distinct, since f is square-free.) Let d_i be the degree of f_i.
(a) Consider the ring

$$A := \mathbb{F}_q[X]/\langle f(X)\rangle.$$

Show that $A \cong \prod_{i=1}^{r} \mathbb{F}_{q^{d_i}}$. **[H]** A is an $\mathbb{F}_q$-vector space of dimension d.
(b) Consider the map $\varphi : A \to A$ that maps $x = X + \langle f(X)\rangle$ to $x^q - x$. Show that φ is an $\mathbb{F}_q$-linear transformation with $\mathrm{Ker}\,\varphi \cong \mathbb{F}_q^r$, and so the nullity of φ equals the number of irreducible factors of f.
(c) Let Q be the matrix of φ with respect to the basis $1, x, \ldots, x^{d-1}$. Describe an algorithm to compute Q. Also design an algorithm to compute a basis of $\mathrm{Ker}\,\varphi$.
(d) Show that if $h \in \mathrm{Ker}\,\varphi$, then

$$f(X) = \prod_{a \in \mathbb{F}_q} \gcd(f(X), h(X) - a).$$

For a suitable $h(X)$, this is a non-trivial factorization of f. This procedure is efficient, when q is small.
(e) Use Berlekamp's method to factor $X^6 + X^5 + X^2 + 1$ over $\mathbb{F}_2$.

*3.6 Arithmetic on Elliptic Curves

The recent popularity of cryptographic systems based on elliptic curve groups over $\mathbb{F}_q$ stems from two considerations. First, discrete logarithms in $\mathbb{F}_q^*$ can be computed in subexponential time. This demands q to be sufficiently large, typically of length 768 bits or more. On the other hand, if the elliptic curve E over $\mathbb{F}_q$ is carefully chosen, the only known algorithms for solving the discrete logarithm problem in $E(\mathbb{F}_q)$ are fully exponential in $\lg q$. As a result, smaller values of q suffice to achieve the desired level of security. In practice, the length of q is required to be between 160 and 400 bits. This leads to *smaller key sizes* for elliptic curve cryptosystems. The second advantage of using elliptic curves is that for a given prime power q, there is only one group $\mathbb{F}_q^*$, whereas there are many elliptic curve groups $E(\mathbb{F}_q)$ (over the same field $\mathbb{F}_q$) with orders ranging from $q+1-2\sqrt{q}$ to $q+1+2\sqrt{q}$. If a particular group $E(\mathbb{F}_q)$ is compromised, we can switch to another curve without changing the base field $\mathbb{F}_q$.

In this section, we start with the description of efficient implementation of the arithmetic in the groups $E(\mathbb{F}_q)$. Then we concentrate on some algorithms for counting the order $\#E(\mathbb{F}_q)$. Knowledge of this order is necessary to find out cryptographically suitable elliptic curves. We consider only prime fields $\mathbb{F}_p$ or fields $\mathbb{F}_{2^n}$ of characteristic 2. So we assume that the curve is defined by Equation (2.8) or Equation (2.9) on p 100 (supersingular curves are not used in cryptography) instead of by the general Weierstrass Equation (2.6) on p 98.

3.6.1 Point Arithmetic

Let us first see how we can efficiently represent points on an elliptic curve E over $\mathbb{F}_q$. Since $P = (h,k) \in E(\mathbb{F}_q)$ corresponds to two elements $h, k \in \mathbb{F}_q$ and since each element of $\mathbb{F}_q$ can be represented using $\leqslant s = \lceil \lg q \rceil$ bits, $2s$ bits suffice to represent P. We can do better than this. Substituting $X = h$ in the equation for E leaves us with a quadratic equation in Y. This equation has two roots of which k is one. If we adopt a convention (for example, see Section 6.2.1) that identifies, using a single bit, which of the two roots the coordinate k is, the storage requirement for P drops to $s+1$ bits. During an on-line computation this compressed representation incurs some overhead and may be avoided. However, for off-line storage and transmission (of public keys, for example), this compression may be helpful.

Explicit formulas for the sum of two points and for the opposite of a point on an elliptic curve E are given in Section 2.11.2. These operations in $E(\mathbb{F}_q)$ can be implemented using a few operations in the ground field $\mathbb{F}_q$.

Computation of mP for $P \in E(\mathbb{F}_q)$ and $m \in \mathbb{N}$ (or, more generally, for $m \in \mathbb{Z}$) can be performed using a repeated-double-and-add algorithm similar to the repeated-square-and-multiply Algorithm 3.9. We leave out the trivial modifications and urge the reader to carry out the details.

Finding a *random* point $P = (h,k) \in E(\mathbb{F}_q)$ is another useful problem. If $q = p$ is an odd prime and we use the short Weierstrass Equation (2.8), we first choose a random $h \in \mathbb{F}_p$ and substitute X by h to get $Y^2 = h^3 + ah + b$. This equation has 2, 0 or 1 solution(s) depending on whether $h^3 + ah + b$ is a quadratic residue or non-residue or 0 modulo p. Quadratic residuosity can be checked by computing the Legendre symbol

(Algorithm 3.15), whereas square roots modulo p can be computed using Tonelli and Shanks' Algorithm 3.16.

For a non-supersingular curve E over $\mathbb{F}_{2^n}$ defined by Equation (2.9), a random point $P = (h, k) \in E(\mathbb{F}_{2^n})$ is chosen by first choosing a random $h \in \mathbb{F}_{2^n}$. Substituting $X = h$ in the defining equation gives $Y^2 + hY + (h^3 + ah^2 + b) = 0$. If $h = 0$, then the unique solution for k is $b^{2^{n-1}}$. If $h \neq 0$, replacing Y by hY and dividing by h^2 transforms the equation to the form $Y^2 + Y + \alpha = 0$ for some $\alpha \in \mathbb{F}_{2^n}$. This equation has two or zero solutions depending on whether the absolute trace $\mathrm{Tr}(\alpha) = \mathrm{Tr}_{\mathbb{F}_{2^n}|\mathbb{F}_2}(\alpha)$ is 0 or 1. If k is a solution, the other solution is $k + 1$. In order to find a solution (if it exists), one may use the (probabilistic) root-finding algorithm of Exercise 3.36. Another possibility is discussed now.

We consider two separate cases. First, if n is odd, then $k := \sum_{i=0}^{(n-1)/2} \alpha^{2^{2i}}$ is a solution, since $\mathrm{Tr}(\alpha) = k^2 + k + \alpha$. On the other hand, if n is even, we first find a $\beta \in \mathbb{F}_{2^n}$ with $\mathrm{Tr}(\beta) = 1$. Since Tr is a homomorphism of the additive groups $(\mathbb{F}_{2^n}, +) \to (\mathbb{F}_2, +)$ and $\mathrm{Tr}(1) = 1$, exactly half of the elements of $\mathbb{F}_{2^n}$ have trace 1. Therefore, a desired β can be quickly found out by selecting elements of $\mathbb{F}_{2^n}$ at random and computing traces of these elements. Now, it is easy to check that $k := \sum_{i=0}^{n-2} \left(\sum_{j=i+1}^{n-1} \beta^{2^j} \right) \alpha^{2^i}$ gives a solution of $Y^2 + Y + \alpha = 0$.

** 3.6.2 Counting Points on Elliptic Curves

Counting points on elliptic curves is a challenging problem, both theoretically and computationally. The first polynomial time (in $\log q$) algorithm invented by Schoof and later made efficient by Elkies and Atkins (and many others), is popularly called the *SEA algorithm*. Unfortunately, even the most efficient implementation of this algorithm is not quite efficient, but it is the only known reasonable strategy, in particular, when $q = p$ is a large (odd) prime of a size of cryptographic interest. The more recent Satoh–FGH algorithm, named after its discoverer Satoh and after Fouquet, Gaudry and Harley who proposed its generalized and efficient versions, is a remarkable breakthrough for the case $q = 2^n$. Both the SEA and the Satoh–FGH algorithms are mathematically quite sophisticated. We now present a brief overview of these algorithms.

The SEA algorithm

We assume that $q = p$ is a large odd prime, this being the typical situation when we apply the SEA algorithm. We also assume that E is given by the short Weierstrass equation $Y^2 = X^3 + aX + b$. Let $q_1 = 2, q_2 = 3, q_3 = 5, \ldots$ be the sequence of prime numbers and t the Frobenius trace of E at p. By Hasse's theorem (Theorem 2.48, p106), $\#E(\mathbb{F}_p) = p + 1 - t$ with $-2\sqrt{p} \leqslant t \leqslant 2\sqrt{p}$. A knowledge of t modulo sufficiently many small primes l allows us to reconstruct t using the Chinese remainder theorem. Because of the Hasse bound on t, it is sufficient to choose l from the primes $q_1, q_2, \ldots$ in succession, until the product $q_1 q_2 \cdots q_r$ exceeds $4\sqrt{p}$. By the prime number theorem (Theorem 2.20, p53), we have $r = \mathrm{O}(\ln p)$ and also $q_i = \mathrm{O}(\ln p)$ for each $i = 1, \ldots, r$.

The most innovative idea of Algorithm 3.31 is the determination of the integers t_i. For $l = q_1 = 2$, the process is easy. We have $t_1 \equiv t \equiv 0 \pmod 2$ if and only if

Algorithm 3.31 **SEA algorithm for elliptic curve point counting**

Input: A prime field $\mathbb{F}_p$, p odd, and an elliptic curve E defined over $\mathbb{F}_p$.
Output: The order of the group $E(\mathbb{F}_p)$.
Steps:
Find (the smallest) $r \in \mathbb{N}$ such that the product $q_1 q_2 \cdots q_r > 4\sqrt{p}$.
for $i = 1, 2, \ldots, r$ { Compute $t_i \in \{0, 1, \ldots, q_i - 1\}$ with $t \equiv t_i \pmod{q_i}$. }
Compute t by combining $t_1, t_2, \ldots, t_r$ using the Chinese Remainder Theorem.

$E(\mathbb{F}_p)$ contains a point of order 2 (a point of the form $(h, 0)$), or equivalently, if and only if the polynomial $X^3 + aX + b$ has a root in $\mathbb{F}_p$. We compute the polynomial gcd $g(X) := \gcd(X^3+aX+b, X^p-X)$ over $\mathbb{F}_p$ and conclude that $t_1 = \begin{cases} 1 & \text{if } g(X) = 1, \\ 0 & \text{otherwise.} \end{cases}$

Determination of t_i for $i > 1$ involves more work. We explain here the original idea due to Schoof. We denote by l the i-th prime q_i and by $E[l] = E_{\bar{\mathbb{F}}_p}[l]$ the set of all l-torsion points of $E(\bar{\mathbb{F}}_p)$ (Definition 2.78, p105). The Frobenius endomorphism $\mathfrak{F} : E(\bar{\mathbb{F}}_p) \to E(\bar{\mathbb{F}}_p)$ that maps $\mathcal{O} \mapsto \mathcal{O}$ and $(h, k) \in E(\bar{\mathbb{F}}_p) \setminus \{\mathcal{O}\}$ to (h^p, k^p) satisfies the relation $\mathfrak{F}^2 - t\mathfrak{F} + p = 0$. If we restrict our attention only to the group $E[l]$, then this relation reduces to $\mathfrak{F}^2 - t_i\mathfrak{F} + p_i = 0$, where $t_i = t \operatorname{rem} l$ and $p_i = p \operatorname{rem} l$, that is, $\mathfrak{F}^2(h, k) - t_i\mathfrak{F}(h, k) + p_i(h, k) = 0$ for all $(h, k) \in E[l]$.

In terms of polynomials, the last relation is equivalent to

$$(X^{p^2}, Y^{p^2}) - t_i(X^p, Y^p) + p_i(X, Y) = 0, \tag{3.4}$$

where the sum and difference follow the formulas for the elliptic curve E. Now, one has to calculate symbolically rather than numerically, since X and Y are indeterminates. These computations can be carried out in the ring $\mathbb{F}_p[X, Y]/\langle f, f_l \rangle$ (instead of in $\mathbb{F}_p[X, Y]$), where $f(X, Y) = Y^2 - (X^3 + aX + b)$ is the defining polynomial of E and $f_l = f_l(X)$ is the l-th division polynomial of E (Section 2.11.2 and Theorem 2.47, p106). Reduction of a polynomial in $\mathbb{F}_p[X, Y]$ modulo f makes its Y-degree $\leqslant 1$, whereas reduction modulo f_l makes the X-degree less than $\deg f_l$ which is $\mathrm{O}(l^2)$. We can try the values $t_i = 0, 1, \ldots, l - 1$ successively until the desired value satisfying Equation (3.4) is found out.

It is not difficult to verify that Schoof's algorithm runs in time $\mathrm{O}(\log^8 p)$ (under standard arithmetic in $\mathbb{F}_p$) and is thus a deterministic polynomial-time algorithm for the point-counting problem. Essentially the same algorithm works for fields $\mathbb{F}_q$ with $q = 2^n$ and has the same running time. Unfortunately, the big exponent (8) in the running time makes Schoof's algorithm quite impractical. Numerous improvements are suggested to bring down this exponent. Elkies and Atkin's modification for the case $q = p$ gives rise to the SEA algorithm which has a running time of $\mathrm{O}(\log^6 p)$ under the standard arithmetic in $\mathbb{F}_p$. This speed-up is achieved by working in the ring $\mathbb{F}_p[X, Y]/\langle f, g_l \rangle$, where g_l is a suitable factor of f_l and has degree $\mathrm{O}(l)$. Couveignes suggests improvements for the fields of characteristic 2. Efficient implementations of the SEA algorithm are reported by Morain, Müller, Dewaghe, Vercauteren and many others. At the time of writing this book, the largest values of q for which the algorithm has been successfully applied are $10^{499} + 153$ (a prime) and 2^{1999} (a power of 2).

The Satoh–FGH algorithm

The Satoh–FGH algorithm is well suited for fields $\mathbb{F}_q$ of small characteristic p and, in particular, for the fields $\mathbb{F}_{2^n}$ of characteristic 2. This algorithm has enabled point-counting over fields as large as $\mathbb{F}_{2^{100002}}$. A generic description of the Satoh–FGH algorithm now follows after the introduction of some mathematical notions. Though our practical interest concentrates on the fields $\mathbb{F}_{2^n}$ only, we consider curves over a general $\mathbb{F}_q$ with $q = p^n$, p a prime.

Recall from Section 2.14 that the ring $\hat{\mathbb{Z}}_p$ of p-adic integers is a discrete valuation ring (Exercises 2.133 and 2.148) with the unique maximal ideal generated by $p := \iota(p) = (p, p, \ldots)$, and the residue field $\hat{\mathbb{Z}}_p/p\hat{\mathbb{Z}}_p$ is isomorphic to $\mathbb{F}_p$.

We represent $\mathbb{F}_q$ as a polynomial algebra over $\mathbb{F}_p$. We analogously define the p-adic ring $\hat{\mathbb{Z}}_q = \hat{\mathbb{Z}}_{p^n} = \hat{\mathbb{Z}}_p[X]/\langle f \rangle$, where f is an irreducible polynomial of degree n in $\hat{\mathbb{Z}}_p[X]$. The elements of $\hat{\mathbb{Z}}_q$ can be viewed as polynomials of degrees $< n$ and with p-adic integers as coefficients. The arithmetic operations in $\hat{\mathbb{Z}}_q$ are polynomial operations in $\hat{\mathbb{Z}}_p[X]$ modulo the defining polynomial f. The ring $\hat{\mathbb{Z}}_p$ is canonically embedded in the ring $\hat{\mathbb{Z}}_q$ (consider constant polynomials).

$\hat{\mathbb{Z}}_q$ turns out to be a discrete valuation ring with maximal ideal $p\hat{\mathbb{Z}}_q$, and the residue field $\hat{\mathbb{Z}}_q/p\hat{\mathbb{Z}}_q$ is isomorphic to $\mathbb{F}_q$.

Definition 3.6 The *projection map* $\pi : \hat{\mathbb{Z}}_p \to \mathbb{F}_p$ is defined as the map that takes a p-adic integer $\alpha = (a_1, a_2, \ldots)$ to $a_1 \in \mathbb{Z}/p\mathbb{Z} = \mathbb{F}_p$, and can be canonically extended to a map $\hat{\mathbb{Z}}_p[X] \to \mathbb{F}_p[X]$ by $\pi(\alpha_0 + \alpha_1 X + \cdots + \alpha_d X^d) := \pi(\alpha_0) + \pi(\alpha_1)X + \cdots + \pi(\alpha_d)X^d$. In particular, this defines a projection map $\pi : \hat{\mathbb{Z}}_q \to \mathbb{F}_q$.

The (*Teichmüller*) *lift* $\omega : \mathbb{F}_q \to \hat{\mathbb{Z}}_q$ is the map that takes $0 \mapsto 0$ and $0 \neq a \mapsto \omega(a)$, where $\omega(a)$ is the unique $(q-1)$-th root of unity in $\hat{\mathbb{Z}}_q$ satisfying $\pi(\omega(a)) = a$ (cf. Exercise 2.160).

The *semi-Witt decomposition* of $\alpha \in \hat{\mathbb{Z}}_q$ is defined to be the unique sequence $a_0, a_1, \ldots$ with $a_i \in \mathbb{F}_q$ such that α has the p-adic expansion $\sum_{i \in \mathbb{N}_0} \omega(a_i) p^i$. □

The p-th power Frobenius endomorphism $\mathfrak{F} : \mathbb{F}_q \to \mathbb{F}_q$, $a \mapsto a^p$, can now be extended to an endomorphism $\mathfrak{G} : \hat{\mathbb{Z}}_q \to \hat{\mathbb{Z}}_q$ as follows. Let $\alpha \in \hat{\mathbb{Z}}_q$ have the semi-Witt decomposition $a_0, a_1, \ldots$ with $a_i \in \mathbb{F}_q$. Then, $\mathfrak{G}(\alpha)$ is the unique element $\beta \in \hat{\mathbb{Z}}_q$ having the semi-Witt decomposition $a_0^p, a_1^p, \ldots$. One can show that $\beta = \mathfrak{G}(\alpha) = \sum_{i \in \mathbb{N}_0} \omega(a_i^p) p^i = \sum_{i \in \mathbb{N}_0} \omega(a_i)^p p^i$. We have $\mathfrak{F}^n = \mathrm{id}_{\mathbb{F}_q}$ and similarly $\mathfrak{G}^n = \mathrm{id}_{\hat{\mathbb{Z}}_q}$.

Now, let $E = E_0$ be an elliptic curve defined over $\mathbb{F}_q$. Application of $\mathfrak{F}$ to the coefficients of E_0 gives another elliptic curve E_1 over $\mathbb{F}_q$ whose rational points are $(\mathfrak{F}(h), \mathfrak{F}(k))$, where $(h, k) \in E(\mathbb{F}_q)$, together with the point at infinity. We may apply $\mathfrak{F}$ to E_1 to get another curve E_2 over $\mathbb{F}_q$ and so on. Since $\mathfrak{F}^n = \mathrm{id}_{\mathbb{F}_q}$, we get a cycle of elliptic curves defined over $\mathbb{F}_q$:

$$E_0 \xrightarrow{\mathfrak{F}} E_1 \xrightarrow{\mathfrak{F}} E_2 \xrightarrow{\mathfrak{F}} \cdots \xrightarrow{\mathfrak{F}} E_{n-1} \xrightarrow{\mathfrak{F}} E_n = E_0 \tag{3.5}$$

Similarly, if $\mathcal{E} = \mathcal{E}_0$ is an elliptic curve defined over $\hat{\mathbb{Z}}_q$, application of $\mathfrak{G}$ leads to a sequence of elliptic curves defined over $\hat{\mathbb{Z}}_q$:

$$\mathcal{E}_0 \xrightarrow{\mathfrak{G}} \mathcal{E}_1 \xrightarrow{\mathfrak{G}} \mathcal{E}_2 \xrightarrow{\mathfrak{G}} \cdots \xrightarrow{\mathfrak{G}} \mathcal{E}_{n-1} \xrightarrow{\mathfrak{G}} \mathcal{E}_n = \mathcal{E}_0 \tag{3.6}$$

We need the *canonical lifting* of an elliptic curve E over $\mathbb{F}_q$ to a curve $\mathcal{E}$ over $\hat{\mathbb{Z}}_q$. Explaining that requires some more mathematical concepts:

Definition 3.7 Let K be a field and let E and E' be two elliptic curves defined over K. A morphism (Definition 2.72, p95) that maps the point $\mathcal{O}$ at infinity of E to the point $\mathcal{O}'$ at infinity of E' is called an *isogeny*. The *zero isogeny* $E \to E'$ maps every point $P \in E(K)$ to $\mathcal{O}'$. A non-zero isogeny is also called a *non-constant isogeny*. Two curves E and E' are called *isogenous*, if there exists a non-constant isogeny $E \to E'$.

The *kernel* $\ker\varphi$ of an isogeny $\varphi : E \to E'$ is defined to be the set $\{P \in E(K) \mid \varphi(P) = \mathcal{O}'\}$. For every non-constant isogeny φ, the kernel $\ker\varphi$ is a *finite* subgroup of $E(K)$.

The set $\mathrm{Hom}(E, E')$ of all isogenies $E \to E'$ is an Abelian group defined as $(\varphi + \psi)(P) = \varphi(P) + \psi(P)$, $\varphi, \psi \in \mathrm{Hom}(E, E')$, $P \in E(K)$. If $E = E'$, then $\mathrm{End}(E) := \mathrm{Hom}(E, E)$ becomes a ring with multiplication defined by composition and is called the *endomorphism ring* of E.

The multiplication-by-m map of E is an isogeny. If $\mathrm{End}(E)$ contains an isogeny not of this type, we call E an elliptic curve with *complex multiplication*. □

Theorem 3.4 For each $i \in \mathbb{N}$, there exists a unique polynomial $\Phi_i(X, Y) \in \mathbb{Z}[X, Y]$ symmetric and of degree $i + 1$ in each of X and Y, such that two curves E and E' (defined over a field K) with j-invariants j and j' satisfy $\Phi_i(j, j') = 0$ if and only if there is an isogeny $E \to E'$ whose kernel is cyclic of order i. ■

Definition 3.8 The polynomials $\Phi_i(X, Y) \in \mathbb{Z}[X, Y]$, $i \in \mathbb{N}$, of Theorem 3.4 are called *modular polynomials*. As an example,

$$\begin{aligned}\Phi_2(X, Y) \;=\; & X^3 + Y^3 - X^2Y^2 + 1488(X^2Y + XY^2) - \\ & 162{,}000(X^2 + Y^2) + 40{,}773{,}375XY + 8{,}748{,}000{,}000(X + Y) - \\ & 157{,}464{,}000{,}000{,}000\,.\end{aligned}$$ □

The next theorem establishes the foundation for lifting curves from $\mathbb{F}_q$ to $\hat{\mathbb{Z}}_q$.

Theorem 3.5 **Lubin–Serre–Tate** Let E be an elliptic curve defined over $\mathbb{F}_q$, $q = p^n$, $p \in \mathbb{P}$, and with j-invariant $j \in \mathbb{F}_q \setminus \mathbb{F}_{p^2}$. There exists an elliptic curve $\mathcal{E}$ defined over $\hat{\mathbb{Z}}_q$ with a unique j-invariant $J \in \hat{\mathbb{Z}}_q$ such that $\Phi_p(J, \mathfrak{G}(J)) = 0$ and $J \equiv j \pmod{p\hat{\mathbb{Z}}_q}$. The curve $\mathcal{E}$ is called the *canonical lift* of E and is unique upto isomorphism. ■

With this definition of lifting of elliptic curves, Cycles (3.5) and (3.6) satisfy the following commutative diagram, where $\mathcal{E}_i$ is the canonical lift of E_i for each $i = 0, 1, \ldots, n$.

$$\begin{array}{ccccccccccc}
\mathcal{E}_0 & \xrightarrow{\mathfrak{G}} & \mathcal{E}_1 & \xrightarrow{\mathfrak{G}} & \mathcal{E}_2 & \xrightarrow{\mathfrak{G}} & \cdots & \xrightarrow{\mathfrak{G}} & \mathcal{E}_{n-1} & \xrightarrow{\mathfrak{G}} & \mathcal{E}_n = \mathcal{E}_0 \\
\downarrow\pi & & \downarrow\pi & & \downarrow\pi & & & & \downarrow\pi & & \downarrow\pi \\
E_0 & \xrightarrow{\mathfrak{F}} & E_1 & \xrightarrow{\mathfrak{F}} & E_2 & \xrightarrow{\mathfrak{F}} & \cdots & \xrightarrow{\mathfrak{F}} & E_{n-1} & \xrightarrow{\mathfrak{F}} & E_n = E_0
\end{array}$$

Algorithm 3.32 outlines the Satoh–FGH algorithm. In order to complete the description of the algorithm, one should specify how to lift curves (that is, a procedural

Algorithm 3.32 **Satoh–FGH algorithm for elliptic curve point counting**

Input: An elliptic curve E over $\mathbb{F}_q$, $q = p^n$, p prime, with j-invariant $\notin \mathbb{F}_{p^2}$.
Output: The cardinality $\#E(\mathbb{F}_q)$ or equivalently the trace $t = q + 1 - \#E(\mathbb{F}_q)$.
Steps:
Compute the curves $E_0, \ldots, E_{n-1}$ and their j-invariants $j_0, \ldots, j_{n-1}$.
Compute the lifted j-invariants $J_0, \ldots, J_{n-1}$.
Compute the lifted curves $\mathcal{E}_0, \ldots, \mathcal{E}_{n-1}$.
Lift the p-torsion groups $E_i[p]$ for $i = 0, \ldots, n-1$.
Compute t and hence $\#E(\mathbb{F}_q)$ from the lifted data.

equivalent of Theorem 3.5) and their p-torsion points and how the lifted data can be used to compute the Frobenius trace t. We leave out the details here.

The elements of $\hat{\mathbb{Z}}_p$ (and hence of $\hat{\mathbb{Z}}_q$) are infinite sequences and hence cannot be represented in computer memory. However, we make an approximate representation by considering only the first m terms of the sequences representing elements of $\hat{\mathbb{Z}}_p$. Working in $\hat{\mathbb{Z}}_p$ with this approximate representation is then essentially the same as working in $\mathbb{Z}/p^m\mathbb{Z}$. For the Satoh–FGH algorithm, we need $m \approx n/2$.

For small p (for example, $p = 2$) and with standard arithmetic in $\mathbb{F}_p$, the Satoh–FGH algorithm has a deterministic running time $\mathrm{O}(n^5)$ and space requirement $\mathrm{O}(n^3)$. With Karatsuba arithmetic the exponent in the running time drops from 5 to nearly 4.17. In addition, this algorithm is significantly easier to implement than optimized versions of the SEA algorithm. These facts are responsible for a superior performance of the Satoh–FGH algorithm over the SEA algorithm (for small p).

3.6.3 Choosing Good Elliptic Curves

Choosing cryptographically suitable elliptic curves is more difficult than choosing good finite fields. First, the order of the elliptic curve group $E(\mathbb{F}_q)$ must have a suitably large prime divisor, say, of bit length 160 or more. In addition, the MOV attack applies to supersingular curves and the anomalous attack to anomalous curves (Definition 2.80 and Section 4.5). So a secure curve must be non-supersingular and non-anomalous. Checking all these criteria for a random curve E over $\mathbb{F}_q$ requires the group order $\#E(\mathbb{F}_q)$. One may use either the SEA algorithm or the Satoh–FGH algorithm to compute $\#E(\mathbb{F}_q)$. Once $\#E(\mathbb{F}_q)$ is known, it is easy to check whether E is supersingular or anomalous. But factoring $\#E(\mathbb{F}_q)$ to find its largest prime divisor may be a difficult task and is not recommended. One may instead extract all the small prime factors of $\#E(\mathbb{F}_q)$ by trial divisions with the primes $q_1 = 2, q_2 = 3, q_3 = 5, \ldots, q_r$ for a predetermined r and write $\#E(\mathbb{F}_q) = m_1 m_2$ where m_1 has all prime factors $\leqslant q_r$ and m_2 has all prime factors $> q_r$. If m_2 is prime and of the desired size, then E is treated as a good curve. Algorithm 3.33 illustrates these steps.

The computation of the group orders $\#E(\mathbb{F}_q)$ takes up most of the execution time of the above algorithm. It is, therefore, of utmost importance to employ good algorithms for point counting. The best algorithms known till date (the SEA and the Satoh–

Algorithm 3.33 **Selecting cryptographically suitable elliptic curves**

Input: A suitably large finite field $\mathbb{F}_q$.
Output: A cryptographically good elliptic curve E over $\mathbb{F}_q$.
Steps:

```
while (1) {
   Generate a random elliptic curve E over F_q.
   Determine #E(F_q).
   if (E is neither supersingular nor anomalous) {
      Try to factorize #E(F_q) using trial division by small primes.
      if (#E(F_q) has a suitably large prime divisor) { Return E }
   }
}
```

FGH algorithms) are only reasonable. Further research in this area may lead to better algorithms in future.

There are ways of generating good curves without requiring the point counting algorithms over large finite fields. One possibility is to use the so-called *subfield curves*. If $\mathbb{F}_q$ has a subfield $\mathbb{F}_{q'}$ of relatively small cardinality, one can choose a random curve E over $\mathbb{F}_{q'}$ and compute $\#E(\mathbb{F}_{q'})$. Since E is also a curve defined over $\mathbb{F}_q$ and $\#E(\mathbb{F}_q)$ can be easily obtained using Theorem 2.51 (p107), we save the lengthy direct computation of $\#E(\mathbb{F}_q)$. However, the drawback of this method is that since E is now chosen with coefficients from a small field $\mathbb{F}_{q'}$, we do not have many choices. The second drawback is that we must have a small divisor q' of q. If q is already a prime, this strategy does not work at all. If $q = p^n$, p a small prime, we need n to have a small divisor n' that corresponds to $q' = p^{n'}$. Sometimes small odd primes p are suggested, but the arithmetic in a non-prime field of some odd characteristic is inherently much slower than that in a field of nearly equal size but of characteristic 2.

Specific curves with complex multiplication (Definition 3.7) over large prime fields have also been suggested in the literature. Finding good curves with complex multiplication involves less computational overhead than Algorithm 3.33, but (like subfield curves) offers limited choice. However, it is important to mention that no special attacks are currently known for subfield curves and also for those chosen by the complex multiplication strategy.

** 3.7 Arithmetic on Hyperelliptic Curves

Let $K = \mathbb{F}_q$ be a finite field and C a hyperelliptic curve of genus g defined over K by Equation (2.13), that is, by

$$C : Y^2 + u(X)Y = v(X)$$

for suitable polynomials $u, v \in K[X]$. We want to implement the arithmetic in the Jacobian $\mathbb{J} = \mathbb{J}(C) = \mathbb{J}_K(C)$. Recall from Section 2.12 that an element of $\mathbb{J}$ can be represented uniquely as a reduced divisor $\mathrm{Div}(a, b)$ for a pair of polynomials $a(x), b(x) \in$

$K[x]$ with a monic, $\deg_x a \leqslant g$, $\deg_x b < \deg_x a$ and $a|(b^2 + bu - v)$. Thus, each element of $\mathbb{J}$ requires $\mathrm{O}(g \log q)$ storage.

3.7.1 Arithmetic in the Jacobian

We first present Algorithm 3.34 that, given two elements $\mathrm{Div}(a_1, b_1), \mathrm{Div}(a_2, b_2)$ of $\mathbb{J}$, computes the reduced divisor $\mathrm{Div}(a, b) \in \mathbb{J}$ which satisfies $\mathrm{Div}(a, b) \sim \mathrm{Div}(a_1, b_1) + \mathrm{Div}(a_2, b_2)$. The algorithm proceeds in two steps:

1. Compute a semi-reduced divisor $\mathrm{Div}(a', b') \sim \mathrm{Div}(a_1, b_1) + \mathrm{Div}(a_2, b_2)$.
2. Compute the reduced divisor $\mathrm{Div}(a, b) \sim \mathrm{Div}(a', b')$.

Both these steps can be performed in (deterministic) polynomial time (in the input size, that is, $g \log q$). Algorithm 3.34 implements the first step and continues to work even when the input divisors are semi-reduced (and not completely reduced).

Algorithm 3.34 **Sum of semi-reduced divisors**

Input: (Semi-)reduced divisors $\mathrm{Div}(a_1, b_1)$ and $\mathrm{Div}(a_2, b_2)$ defined over K.
Output: A semi-reduced divisor $\mathrm{Div}(a', b') \sim \mathrm{Div}(a_1, b_1) + \mathrm{Div}(a_2, b_2)$.
Steps:

$d_1 := \gcd(a_1, a_2) = u_1 a_1 + u_2 a_2$. /* Extended gcd in $K[X]$ */
$d_2 := \gcd(d_1, b_1 + b_2 + u) = v_1 d_1 + v_2(b_1 + b_2 + u)$. /* Extended gcd in $K[X]$ */

$$a' := \left(\frac{a_1 a_2}{d_2^2} \right).$$

$$b' := \left(\frac{u_1 v_1 a_1 b_2 + u_2 v_1 a_2 b_1 + v_2(b_1 b_2 + v)}{d_2} \right) \mathrm{rem}\ a'.$$

It is an easy check that the two expressions appearing between pairs of big parentheses in Algorithm 3.34 are polynomials. This algorithm does only a few gcd calculations and some elementary arithmetic operations on polynomials of $K[X]$. If the input polynomials (a_1, a_2, b_1, b_2) correspond to reduced divisors, then their degrees are $\leqslant g$ and hence this algorithm runs in polynomial time in the input size. Furthermore, in that case, the output polynomials a' and b' are of degrees $\leqslant 2g$.

We now want to compute the unique reduced divisor $\mathrm{Div}(a, b)$ equivalent to the semi-reduced divisor $\mathrm{Div}(a', b')$. This can be performed using Algorithm 3.35. If the degrees of the input polynomials a' and b' are $\mathrm{O}(g)$ (as is the case with those output by Algorithm 3.34), Algorithm 3.35 takes a time polynomial in $g \log q$. To sum up, two elements of $\mathbb{J}$ can be added in polynomial time. The correctness of the two algorithms is not difficult to establish, but the proof is long and involved and hence omitted. Interested readers might look at the appendix of Koblitz's book [154].

For an element $\alpha \in \mathbb{J}$ and $n \in \mathbb{N}$, one can easily write an algorithm (similar to Algorithm 3.9) to compute $n\alpha$ using $\mathrm{O}(\log n)$ additions and doublings in $\mathbb{J}$.

3.7.2 Counting Points in Jacobians of Hyperelliptic Curves

For a hyperelliptic curve C of genus g defined over a field $\mathbb{F}_q$, we are interested in the order of the Jacobian $\mathbb{J} = \mathbb{J}(C) = \mathbb{J}_{\mathbb{F}_q}(C)$ rather than in the cardinality of the curve

Algorithm 3.35 **Reduction of a semi-reduced divisor**

Input: A semi-reduced divisor $\mathrm{Div}(a', b')$ defined over K.
Output: The reduced divisor $\mathrm{Div}(a, b) \sim \mathrm{Div}(a', b')$.
Steps:
$(a, b) := (a', b')$.
while $(\deg a > g)$ {
  $a' := -\left(\dfrac{b^2 + bu - v}{a}\right)$. /* a' is a polynomial */
  $b' := -(u + b) \bmod a'$.
  $(a, b) := (a', b')$.
}
$a := [\mathrm{lc}(a)]^{-1} a$. /* Make a monic */

$C(\mathbb{F}_q)$. Algorithmic and implementational studies of counting $\#\mathbb{J}$ have not received enough research endeavour till date and though polynomial-time algorithms are known to this effect (at least for curves of small genus), these algorithms are far from practical for hyperelliptic curves of cryptographic sizes. In this section, we look at some of these algorithms.

We start with some theoretical results which are generalizations of those for elliptic curves. The Frobenius endomorphism $\mathfrak{F} : \bar{\mathbb{F}}_q \to \bar{\mathbb{F}}_q$, $x \mapsto x^q$, is a (non-trivial) $\mathbb{F}_q$-automorphism of $\bar{\mathbb{F}}_q$. The map $\mathfrak{F}$ naturally (that is, coordinate-wise) extends to the points on $C(\bar{\mathbb{F}}_q)$ and also to divisors and, in particular, to the Jacobian $\mathbb{J}_{\bar{\mathbb{F}}_q}(C)$ as well as to $\mathbb{J} = \mathbb{J}_{\mathbb{F}_q}(C)$. For a reduced divisor $\mathrm{Div}(a, b) \in \mathbb{J}$, we have $\mathfrak{F}(\mathrm{Div}(a, b)) = \mathrm{Div}(\mathfrak{F}(a), \mathfrak{F}(b))$, where for a polynomial $h \in \bar{\mathbb{F}}_q[X]$ the polynomial $\mathfrak{F}(h)$ is obtained by applying the map $\mathfrak{F}$ to the coefficients of h. It is known that $\mathfrak{F}$ satisfies a monic polynomial $\chi(X)$ of degree $2g$ with integer coefficients. For example, for $g = 1$ (elliptic curves) we have

$$\chi(X) = X^2 - tX + q,$$

where t is the trace of Frobenius at q. For $g = 2$, we have

$$\chi(X) = X^4 - t_1 X^3 + t_2 X^2 - t_1 q X + q^2 \tag{3.7}$$

for integers t_1, t_2. The cardinality $\#\mathbb{J}$ is related to the polynomial $\chi(X)$ as

$$n := \#\mathbb{J}_{\mathbb{F}_q}(C) = \chi(1)\,,$$

and satisfies the inequalities

$$\left\lceil (\sqrt{q} - 1)^{2g} \right\rceil \leqslant n \leqslant \left\lfloor (\sqrt{q} + 1)^{2g} \right\rfloor. \tag{3.8}$$

Thus n lies in a rather narrow interval, called the *Hasse–Weil interval*, of width $w := \left\lfloor (\sqrt{q} + 1)^{2g} \right\rfloor - \left\lceil (\sqrt{q} - 1)^{2g} \right\rceil \approx 4gq^{g-\frac{1}{2}}$,

Theorem 2.50 can be generalized as follows:

Theorem 3.6 **Structure theorem for $\mathbb{J}_{\mathbb{F}_q}(C)$** The Jacobian $\mathbb{J} = \mathbb{J}_{\mathbb{F}_q}(C)$ is the direct sum of at most $2g$ cyclic groups, that is, $\mathbb{J} \cong \mathbb{Z}_{n_1} \oplus \mathbb{Z}_{n_2} \oplus \cdots \oplus \mathbb{Z}_{n_r}$ with $r \leqslant 2g$, $n_1, \ldots, n_r \geqslant 2$ and $n_{i+1} | n_i$ for each $i = 1, 2, \ldots, r - 1$. ■

The exponent of $\mathbb{J}$ (See Exercise 3.42) is clearly $m := \operatorname{Exp}\mathbb{J} = n_1$. Since $m|n$, there are $\leqslant \lceil (w+1)/m \rceil$ possibilities for n for a given m (where w is the width of the Hasse–Weil interval). In particular, n is uniquely determined by m, if $m > w$. From the Hasse–Weil bound, we have $(\sqrt{q}-1)^{2g} \leqslant \lceil (\sqrt{q}-1)^{2g} \rceil \leqslant n = n_1 \cdots n_r \leqslant n_1^r \leqslant n_1^{2g} = m^{2g}$, that is, $m \geqslant \sqrt{q}-1$. There are examples with $m = \sqrt{q}-1$. On the other hand, $w \approx 4gq^{g-\frac{1}{2}}$. So it is possible to have $m \leqslant w$, though such curves are relatively rare. In the more frequent case ($m > w$), Algorithm 3.36 determines n.

Algorithm 3.36 **Hyperelliptic curve point counting**

Input: A hyperelliptic curve C of genus g defined over $\mathbb{F}_q$.
Output: The cardinality n of the Jacobian $\mathbb{J}_{\mathbb{F}_q}(C)$.
Steps:
$m := 1$.
while ($m \leqslant w$) {
  Choose a random element $x \in \mathbb{J}$.
  Determine $\nu := \operatorname{ord} x$.
  $m := \operatorname{lcm}(m, \nu)$.
}
$n :=$ the unique multiple of m in the Hasse–Weil interval.

Since $\operatorname{Exp}\mathbb{J} = \operatorname{lcm}(\operatorname{ord} x \mid x \in \mathbb{J})$, the above algorithm eventually (in practice, after few executions of the while loop) computes this exponent. However, if $\operatorname{Exp}\mathbb{J} \leqslant w$, the algorithm never terminates. Thus, we may forcibly terminate the algorithm by reporting *failure*, after sufficiently many random elements x are tried (and we continue to have $m \leqslant w$). In order to complete the description of the algorithm, we must specify a strategy to compute $\nu := \operatorname{ord} x$ for a randomly chosen $x \in \mathbb{J}$. Instead of computing ν directly, we compute an (integral) multiple μ of ν, factorize μ and then determine ν. Since $nx = 0$, we search for a desired multiple μ in the Hasse–Weil interval. This search can be carried out using a baby-step–giant-step (Section 4.4) or a birthday-paradox (Exercise 2.172) method, and the algorithm achieves an expected running-time of $\tilde{\mathrm{O}}(\sqrt{w})$ which is exponential in the input size. This method, therefore, cannot be used except when n is small.

For hyperelliptic curves of small genus g, generalizations of Schoof's algorithm (Algorithm 3.31) can be used. Gaudry and Harley [106] describe the case $g = 2$. One computes the polynomial $\chi(X)$ of Equation (3.7), that is, the values of t_1 and t_2 modulo sufficiently many small primes l. Since the roots of $\chi(X)$ are of absolute value $\sqrt{q}$, we have $|t_1| \leqslant 4\sqrt{q}$ and $|t_2| \leqslant 6q$. Therefore, determination of t_1 and t_2 modulo $\mathrm{O}(\log q)$ small primes l uniquely determines $\chi(X)$ (as well as $n = \chi(1)$).

Let $\mathbb{J}[l] := \{x \in \mathbb{J} \mid lx = 0\}$ be the set of l-torsion points of $\mathbb{J}$. The Frobenius map restricted to $\mathbb{J}[l]$ satisfies

$$\mathfrak{F}^4(D) - t_{1,l}\mathfrak{F}^3(D) + t_{2,l}\mathfrak{F}^2(D) - t_{1,l}q_l\mathfrak{F}(D) + q_l^2 D = 0\,, \qquad \textbf{(3.9)}$$

where $t_{1,l} := t_1 \operatorname{rem} l$, $t_{2,l} := t_2 \operatorname{rem} l$ and $q_l := q \operatorname{rem} l$. By exhaustively trying all (that is, $\leqslant l^2$) possibilities for $t_{1,l}$ and $t_{2,l}$, one can find out their actual values, that is, those values that cause the left side of Equation (3.9) to vanish (symbolically).

A result by Kampkötter [144] allows us to consider only the reduced divisors $D \in \mathbb{J}[l]$ of the form $D = \mathrm{Div}(a, b)$ with $a(X) = X^2 + a_1X + a_0$ and $b(X) = b_1X + b_0$. There exists an ideal $\mathfrak{a}_l$ of the polynomial ring $\mathbb{F}_q[A_1, A_0, B_1, B_0]$ such that a reduced divisor D of this special form lies in $\mathbb{J}[l]$ if and only if $f(a_1, a_0, b_1, b_0) = 0$ for all $f \in \mathfrak{a}_l$. Thus the computation of the left side of Equation (3.9) may be carried out in the ring $\mathbb{F}_q[A_1, A_0, B_1, B_0]/\mathfrak{a}_l$. An explicit set of generators for $\mathfrak{a}_l$ can be found in Kampkötter [144]. To sum up, we get a polynomial-time algorithm.

Working (modulo $\mathfrak{a}_l$) in the 4-variate polynomial ring $\mathbb{F}_q[A_1, A_0, B_1, B_0]$ is, indeed, expensive. Use of *Cantor's division polynomials* [43] essentially reduces the arithmetic to proceed with a single variable (instead of four). We do not explore further along this line, but only mention that for $g = 2$ Schoof's algorithm employing division polynomials runs in time $\mathrm{O}(\log^9 q)$. Although this is a theoretical breakthrough, the prohibitively large exponent (9) in the running-time precludes the feasibility of using the algorithm in the range of interest in cryptography.

EXERCISE SET 3.7

3.42 Let G be a multiplicative group (not necessarily Abelian and/or finite) with identity e. Let $S := \{n \in \mathbb{Z} \mid g^n = e \text{ for all } g \in G\} \subseteq \mathbb{Z}$.
(a) Show that S is a subgroup of $\mathbb{Z} = (\mathbb{Z}, +)$.
(b) Show that every subgroup of $\mathbb{Z}$ is generated by a single element. In particular, $S = \langle m \rangle$ for some integer m. Without loss of generality, we can take $m \geqslant 0$. This m is called the *exponent* of the group G and is denoted by $\mathrm{Exp}\, G$.
(c) If G is finite, show that $\mathrm{Exp}\, G | \operatorname{ord} G$.
(d) If G is finite and Abelian, show that $\mathrm{Exp}\, G = \mathrm{lcm}(\operatorname{ord} g \mid g \in G)$. Deduce that in this case there exists $x \in G$ such that $\operatorname{ord} x = \mathrm{Exp}\, G$.

3.8 Random Numbers

So far we have met several situations where we needed random elements from a (finite) set, for example, the set $\mathbb{Z}_n$ (or $\mathbb{Z}_n^*$) or the set $\mathbb{F}_q$ (or $\mathbb{F}_q^*$) or the set $E(\mathbb{F}_q)$ of $\mathbb{F}_q$-rational points on an elliptic (or hyperelliptic) curve. By *randomness*, we here mean that each element $a \in S$ is equally likely to get selected, that is, if $\#S = n$, then each element of S is selected with probability $1/n$. Since elements of a set S of cardinality n can be represented as bit strings of length $\leqslant \lceil \lg(n + 1) \rceil$, the problem of selecting a random element of S essentially reduces to the problem of generating (finite) random sequences of bits. A random sequence of bits is one in which every bit has a probability of $1/2$ of being either 0 or 1 (irrespective of the other bits in the sequence).

3.8.1 Pseudorandom Bit Generators

Generating a (truly) random sequence of bits seems to be an impossible task. Some natural phenomena, such as electronic noise from a specifically designed integrated circuit, can be used to generate random bit sequences. However, such systems are

prone to malfunctioning, often influenced by observations and are, of course, costly. A software solution is definitely the more practical alternative. Phenomena, like the system clock, the work load or memory usage of a machine, that can be captured by programs may be used to generate random bit sequences. But this strategy also suffers from various drawbacks. First of all the sequences generated by these methods would not be (truly) random. Moreover they are vulnerable to attacks by adversaries (for example, if a random bit generator is based on the system clock and if the adversary knows the approximate time when a bit sequence is generated using that generator, she will have to try only a few possibilities to generate the same sequence).

In order to obviate these difficulties, *pseudorandom bit generators* (PRBG) are commonly used. A bit string $a_0a_1a_2\ldots$ is generated by a PRBG following a specific strategy, which is more often that not a (mathematical) algorithm. The first bit a_0 is based on certain initial value, called a *seed*, whereas for $i \geqslant 1$the bit a_i is generated as a predetermined function of some or all of the previous bits $a_0, \ldots, a_{i-1}$. Since the resulting bit a_i is now functionally dependent on the previous bits, the sequence is not at all random (but deterministic), but we are happy if the sequence $a_0a_1a_2\ldots$ *looks* or *behaves* random. A random behaviour of a sequence is often examined by certain well-known statistical tests. If a generator generates bit sequences passed by these tests, we call it a PRBG and sequences available from such a generator *pseudorandom bit sequences*. Various kinds of PRBGs are used for generating pseudorandom bit sequences. We won't describe them here, but concentrate on a particular kind of generators that has a special significance in cryptography.

3.8.2 Cryptographically Strong Pseudorandom Bit Generators

A PRBG for which no polynomial time algorithms exist (provably or not) in order to predict with probability significantly larger than $1/2$ a bit in a sequence generated by the PRBG from a knowledge of the previous bits (but without the knowledge of the seed) is called a *cryptographically strong* (or *secure*) *pseudorandom bit generator* or a CSPRBG in short. Usually, an intractable computational problem (see Section 4.2) is at the heart of the security of a CSPRBG. As an example, we now explain the *Blum–Blum–Shub* (or *BBS*) generator.

Algorithm 3.37 **Blum–Blum–Shub pseudorandom bit generator**

Input: $m \in \mathbb{N}$.
Output: A cryptographically strong pseudorandom bit sequence $a_0a_1a_2\ldots$.
Steps:
Generate two (distinct) large primes p and q each $\equiv 3 \pmod 4$.
$n := pq$.
Generate a (random) seed $s \in \mathbb{Z}_n^*$.
$x_0 := s^2 \pmod n$.
for $i = 0, \ldots, m$ {
  $a_i :=$ the least significant bit of x_i.
  $x_{i+1} := x_i^2 \pmod n$.
}

In Algorithm 3.37, we have used indices for the sequence x_i for the sake of clarity. In an actual implementation, all indices may be removed, that is, one may use a single variable x to store and update the sequence x_i. Furthermore, if there is no harm in altering the value of s, one might even use the same variable for s and x.

The cryptographic security of the BBS generator stems from the presumed intractability of factoring integers or of computing square roots modulo a composite integer (here $n = pq$) (see Exercise 3.43). Note that p, q and s have to be kept secret, whereas n can be made public. A knowledge of x_{m+1} is also not expected to help an opponent and may too be made public. For achieving the desired level of secrecy, p and q should be of nearly equal size and the size of n should be sufficiently large (say, 768 bits or more). Generating each bit by the BBS generator involves a modular squaring and is, therefore, somewhat slow (compared to the traditional PRBGs which do not guarantee cryptographic security). However, the BBS generator can be used for moderately infrequent purposes, for example, for the generation of a session key. Moreover, a maximum of $\lg \lg n$ (least significant) bits (instead of 1 as in the above snippet) can be extracted from each x_i without degrading the security of the generator.

It is evident that any (infinite) sequence $a_0 a_1 \cdots$ generated by the BBS generator must be periodic. As an extreme example, if $s = 1$, then the BBS generator outputs a sequence of one-bits only. We are interested in rather short (sub)sequences (of such infinite sequences). Therefore, it suffices if the length of the period is reasonably large (for a random seed s). This is guaranteed if one uses strong primes (Definition 3.5)

3.8.3 Seeding Pseudorandom Bit Generators

The way we have defined PRBG (or CSPRBG) makes it evident that the unpredictability of a pseudorandom bit sequence essentially reduces to that of the seed. Care should, therefore, be taken in order to choose the values of the seed. The seed need not be randomly or pseudorandomly generated, but should have a high degree of unpredictability, so that it is infeasible for an adversary to have a reasonably quick guess of it. As an example, assume that we intend to generate a suitable seed s for the BBS generator with a 1024-bit modulus n. If we employ for that purpose a specific algorithm (known to the opponent) using *only* the built-in random number generator of a standard compiler and if this built-in generator has a 32-bit seed σ, then there are only 2^{32} possibilities for s, even when s itself is 1024 bits long. Thus an adversary has to try at most 2^{32} (2^{31} on an average) values of σ in order to guess the correct value of s. So we must add further unpredictability to the resulting seed value s. This can be done by setting the bits of s depending on several factors, like the system clock, the system load, the memory usage, keyboard inputs from a human user and so on. Each of such factors might not be individually completely unpredictable, but their combined effect should preclude the feasibility of an exhaustive search by the opponent. After all, we have 1024 bits of s to fill up and even if the total search space of possible values of s is as low as 2^{160}, it would be impossible for the opponent to guess s in a reasonable span of time. Note that more often than not the values of the seed need not be remembered: that is, need not be regenerated afterwards. As a result, there is no harm in introducing unpredictability in s caused by certain factors that we would not ourselves be able to reproduce in future.

EXERCISE SET 3.8

3.43 With the notations of Algorithm 3.37 show that:

(a) Every quadratic residue $x \in \mathbb{Z}_n^*$ has four distinct square roots modulo n, of which exactly one, say y, is a quadratic residue modulo n. **[H]**

(b) The square root y of x can be obtained by solving the simultaneous congruences $y \equiv x^{(p+1)/4} \pmod{p}$ and $y \equiv x^{(q+1)/4} \pmod{q}$.

(c) The bit sequence $a_0 a_1 \ldots a_m$ is uniquely determined by (n and) x_{m+1}.

(d) One can compute in polynomial (in $\log n$ and m) time the bit sequence $a_0 a_1 \ldots a_m$ from the knowledge of n and x_{m+1}, if either

(1) the primes p and q are known, or

(2) one can check in polynomial (in $\log n$) time if an arbitrary element $y \in \mathbb{Z}_n^*$ is a quadratic residue modulo n and if so, compute in polynomial time the square roots of y modulo n.

Chapter Summary

This chapter deals with the algorithmic details needed for setting up public-key cryptosystems. We study algorithms for selecting public-key parameters and for carrying out the basic cryptographic primitives. Algorithms required for cryptanalysis are dealt with in Chapters 4 and 7.

We start the chapter with a discussion on algorithms. *Time* and *space complexities* of algorithms are discussed first and the standard *order notations* are explained. Next we study the class of *randomized algorithms* which provide practical solutions to many computational problems that do not have known efficient *deterministic algorithms*. In the worst case, a randomized algorithm may take exponential running time and/or may output an incorrect answer. However, the probability of these bad behaviours of a randomized algorithm can be made arbitrarily low. We finally discuss *reduction* between computational problems. A reduction helps us conclude about the complexity of one problem relative to that for another problem.

Many popular public-key cryptosystems are based on working modulo big integers. These integers have sizes up to several thousand bits. One can not represent such integers with full precision by built-in data types supplied by common programming languages. So we require efficient ways of representing and doing arithmetic on big integers. We carefully deal with the implementation of the *arithmetic* on *multiple-precision integers*. We provide a special treatment of computation of *gcd*'s and *extended gcd*'s of integers. We utilize these arithmetic functions in order to implement *modular arithmetic*. Most public-key primitives involve *modular exponentiations* as the most time-consuming steps. In addition to the standard *square-and-multiply algorithm*, certain special tricks (including *Montgomery exponentiation*) that help speed up modular exponentiation are described at length in this section.

In the next section, we deal with some other number-theoretic algorithms. One important topic is the determination of whether a given integer is prime. The *Miller–Rabin primality test* is an efficient algorithm for *primality testing*. This algorithm is, however, randomized in the sense that it may declare some composite integers as primes. Using suitable choices of the relevant parameters, the probability of this error may be reduced to very low values ($\leqslant 2^{-80}$). We also briefly introduce the deterministic polynomial-time *AKS algorithm* for primality testing. Since we can easily check for the primality of integers, we can generate random primes by essentially searching in a pool of randomly generated odd integers of a given size. Security in some cryptosystems require such random primes to possess some special properties. We present *Gordon's algorithm* for generating cryptographically *strong primes*. The section ends with a study of the *Tonelli–Shanks algorithm* for computing square roots modulo a big prime.

Next, we concentrate on the implementation of the *finite field arithmetic*. The arithmetic of a field of prime cardinality p is the same as integer arithmetic modulo p and is discussed in detail earlier. The other finite fields that are of interest to cryptology are extension *fields of characteristic* 2. In order to study the arithmetic in these fields, one first requires arithmetic of the *polynomial ring* $\mathbb{Z}_2[X]$. We discuss the basic operations in this ring. Next we talk about algorithms for *checking irreducibility* of polynomials and for obtaining (random) irreducible polynomials in $\mathbb{Z}_2[X]$. If $f(X)$ is such a polynomial of degree d, the arithmetic of the field $\mathbb{F}_{2^d}$ is the same as the

arithmetic of $\mathbb{Z}_2[X]$ modulo the *defining polynomial* $f(X)$. In order that a finite field $\mathbb{F}_q$ is cryptographically safe, we require $q-1$ to have a prime factor of sufficiently big size (160 bits or more). Suppose that the factorization of $q-1$ is provided. We discuss algorithms that compute the *order* of elements in $\mathbb{F}_q^*$, that check if a given element is a *generator* of the cyclic group $\mathbb{F}_q^*$, and that produce random generators of $\mathbb{F}_q^*$. We end the study of finite fields by discussing a way to *factor polynomials* over finite fields. The standard algorithm comprising the three steps *square-free factorization*, *distinct-degree factorization* and *equal-degree factorization* is explained in detail. The exercises cover the details of an algorithm to compute the *roots of polynomials* over finite fields.

The *arithmetic of elliptic curves* over finite fields is dealt with next. Each operation in the elliptic curve group can be realized by a sequence of operations over the underlying field. The multiple of a point on an elliptic curve can be computed by a repeated *double-and-add algorithm* which is the same as the square-and-multiply algorithm for modular exponentiation, applied to an additive setting. We also discuss ways of selecting *random points* on elliptic curves. We then present two algorithms for *counting points* in an elliptic curve group. The *SEA algorithm* is suitable for curves over prime fields, whereas the *Satoh–FGH algorithm* works efficiently for curves over fields of characteristic 2. Once we can determine the order of an elliptic curve group, we can choose good elliptic curves for cryptographic usage.

In the next section, we study the *arithmetic of hyperelliptic curves*. We describe ways to represent elements of the Jacobian by pairs of polynomials and to do arithmetic on elements in this representation. We also discuss two algorithms for *counting points* in a Jacobian.

In the last section, we address the issue of generation of pseudorandom bits. We define the concept of *cryptographically strong pseudorandom bit generator* and provide an example, namely the *Blum–Blum–Shub generator*, which is cryptographically strong under the assumption that taking square roots modulo a big composite integer is computationally intractable.

Suggestions for Further Reading

The basic algorithmic issues discussed in Section 3.2 can be found in any text-book on data structure and algorithms. One can, for example, look at [7, 8, 61]. However, most of these elementary books do not talk about randomization and parallelization issues. We refer to [214] for a recent treatise on randomized algorithm. Also see Rabin's papers [247, 248].

Complexity theory deals with classifying computational problems based on the known algorithms for solving them and on reduction of one problem to another. A simple introduction to complexity theory is the book [280] by Sipser. Chapter 2 of Koblitz's book [154] is also a compact introduction to computational complexity meant for cryptographers. Also see [113].

Knuth's book [147] is seemingly the best resource to look at for a comprehensive treatment on multiple-precision integer arithmetic. The proofs of correctness of many algorithms, that we omitted in Section 3.3, can be obtained in this book. This can be supplemented by the more advanced algorithms and important practical tips compiled in the book [56] by Cohen who designed a versatile computational number theory package known as PARI. Montgomery's multiplication algorithm appeared in [210]. Also see Chapter 14 of Menezes et al. [194] for more algorithms and implementation issues.

Most of the important papers on primality testing [3, 4, 5, 116, 175, 204, 248, 287] have been referred in Section 3.4.1. Also see the survey [164] due to Lenstra and Lenstra. Gordon's algorithm for generating strong primes appeared in [118]. The book [69] by Crandall and Pomerance is an interesting treatise on prime numbers, written with a computational perspective. The modular square-root Algorithm 3.16 is essentially due to Tonelli (1891). Algebraic number theory is treated from a computation perspective in Cohen [56] and Pohst and Zassenhaus [235].

Arithmetic on finite fields is discussed in many books including [179, 191]. Finite fields find recent applications in cryptography and coding theory and as such it is necessary to have efficient software and hardware implementations of finite field arithmetic. A huge number of papers have appeared in the last two decades, that talk about these implementation issues. Chapter 5 of Menezes [191] talks about optimal normal bases (Section 2.9.3 of the current book) which speeds up exponentiation in finite fields.

Factoring univariate polynomials over finite fields is a topic that has attracted a lot of research attention. Berlekamp's Q-matrix method [21] is the first modern algorithm for this purpose. Computationally efficient versions of the algorithm discussed in Section 3.5.4 have been presented in Gathen and Shoup [104] and Kaltofen and Shoup [143]. The best-known running time for a *deterministic* algorithm for univariate factorization over finite fields is due to Shoup [272]. Shparlinski shows [274] that Shoup's algorithm on a polynomial in $\mathbb{F}_q[X]$ of degree d uses $O(q^{1/2}(\log q)d^{2+\epsilon})$ bit operations. This is fully exponential in $\log q$.

The book [103] by von zur Gathen and Gerhard is a detailed treatise on many topics discussed in Sections 3.2 to 3.5 of the current book. Mignotte's book [203] and the one by [108] by Geddes et al. also have interesting coverage. Also see Chapter 1 of Das [72] for a survey of algorithms for various computational problems on finite fields.

For elliptic curve arithmetic, look at Blake et al. [24], Hankerson et al. [123] and Menezes [192]. The first polynomial-time algorithm for counting points in elliptic curves over a finite field $\mathbb{F}_q$ has been proposed by Schoof. The original version of this

algorithm runs in time $O(\log^8 q)$. Later Elkies improved this running time to $O(\log^6 q)$ for most of the elliptic curves. Further modifications due to Atkin gave rise to what we call the SEA algorithm. Schoof's paper [264] talks about this point-counting algorithm and includes the modifications due to Elkies and Atkin. Also look at the article [85] by Elkies.

The Satoh–FGH algorithm is originally due to Satoh [256]. Fouquet et al. [94] have proposed a modification of Satoh's algorithm to work for fields of characteristic 2. They also report large-scale implementations of the modified algorithm. Also see Fouquet et al. [95] and Skjernaa [281].

Recently, there has been lot of progress in point counting algorithms, in particular, for fields of characteristic 2. The most recent account of this can be found in Lercier and Lubicz [177]. The authors of this paper later reported implementation of their algorithm for counting points in an elliptic curve over $\mathbb{F}_{2^{100002}}$. This computation took nearly 82 hours on a 731 MHz Alpha EV6 processor. With these new developments, the point counting problem is practically solved for fields of small characteristics. However, for prime fields the known algorithms require further enhancements in order to be useful on a wide scale.

Finding good random elliptic curves for cryptographic purposes has also been an area of active research recently. With the current status of solving the elliptic curve discrete-log problem, the strategy we mentioned in Algorithm 3.33 is quite acceptable as long as good point-counting algorithms are at our disposal (they are now). For further discussions on this topic, we refer the reader to two papers [95, 176].

The appendix in Koblitz's book [154] is seemingly the best source for learning hyperelliptic curve arithmetic. This is also available as a CACR technical report [195]. Gaudry and Harley's paper [106] has more on the hyperelliptic curve point-counting algorithms we discussed in Section 3.7.2. Hess et al. [126] discuss methods for computing hyperelliptic curves for cryptographic usage.

Chapter 5 of Menezes et al. [194] is devoted to the generation of pseudorandom bits and sequences. This chapter lists the statistical tests for checking the randomness of a bit sequence. It also describes two cryptographically secure pseudorandom bit generators other than the BBS generator (Algorithm 3.37). The BBS generator was originally proposed by Blum et al. [26]. Also see Chapter 3 of Knuth [147].

This page is intentionally left blank

The Intractable Mathematical Problems

4

4.1 Introduction
4.2 The Problems at a Glance
4.3 The Integer Factorization Problem
4.4 The Finite Field Discrete Logarithm Problem
4.5 The Elliptic Curve Discrete Logarithm Problem
4.6 The Hyperelliptic Curve Discrete Logarithm Problem
4.7 Solving Large Sparse Linear Systems over Finite Rings
4.8 The Subset Sum Problem
Chapter Summary
Sugestions for Further Reading

It is insufficient to protect ourselves with laws;
we need to protect ourselves with mathematics.
— Bruce Schneier

Most number theorists considered the small group of colleagues
that occupied themselves with these problems
as being inflicted with an incurable but harmless obsession.
— Arjen K. Lenstra and Hendrik W. Lenstra, Jr. [164]

All mathematics is divided into three parts:
cryptography (paid for by CIA, KGB and the like),
hydrodynamics (supported by manufacturers of atomic submarines) and
celestial mechanics (financed by military and other institutions
dealing with missiles, such as NASA).
— V. I. Arnold [13]

4.1 Introduction

Public-key cryptographic systems are based on the *apparent* intractability of solving certain computational problems. However, there is very little evidence (if any) to corroborate the fact that algorithmic solutions to these problems are really very difficult. In spite of intensive studies over a long period, mathematicians and cryptologists have not come up with good algorithms, and it is their failures that justify the attempts to go on building *secure* cryptographic protocols based on these problems. The inherent assumption is that it would be infeasible for an opponent having practical amounts of computing resources to break these cryptosystems in a reasonable amount of time. Of course, the fear remains that someone may devise a fast algorithm and our cryptosystems may not pass the security guarantees. On the other extreme, it is also possible that someone proves the theoretical (and, hence, practical) impossibility of solving such a problem in a *small* (like polynomial) amount of time, and our cryptosystems become secure for ever (well, at least until other paradigms of computing, like the yet practically non-implementable quantum computing, solve the problems efficiently).

Whether you are a cryptographer or a cryptanalyst, it is important, if not essential, to be aware of the best methods available till date to attack the intractable problems of cryptography. In the first place, this knowledge quantifies practical security margins of the protocols, for instance, by dictating the determination of the input sizes as a function of the security requirements. Let us take a specific example: With today's computing power and known integer factorization algorithms, we assert that a message that needs to be kept secret for a day or two may be encrypted by a 768-bit RSA key, whereas if one wants to maintain the security for a year or more, much longer keys are needed. The second point in studying the known cryptanalytic algorithms is that though general-purpose algorithms for solving these problems are still unknown, there are good algorithms for specific cases—the cases to be avoided by the designers of cryptographic applications. For example, there is a linear-time algorithm to attack cryptographic systems based on anomalous elliptic curves. The moral is that one must not employ these curves for cryptographic applications. The third reason for studying cryptanalytic algorithms is sentimental. The fact that we are still unable to answer some simply stated questions even after spending a reasonable amount of collective effort is indeed humbling. To worsen matters, cryptography thrives by exploiting this scientific inadequacy. Cryptanalysis, though seemingly unlawful from a cryptographer's viewpoint, turns out to be a deep and beautiful area of applied mathematics. Ironically enough, it is quite common that the proponents of cryptographic protocols are themselves most interested to see the end. The journey goes on... Read on!

It may appear somewhat unusual to discuss the cryptanalytic algorithms prior to the cryptographic ones (see Chapter 5). We find this order convenient in that one must first know the intractable problems before applying them in cryptographic protocols. Moreover, the known attacks help one fix the parameters for use in the cryptographic algorithms. We defer till Chapter 7 other cryptanalytic techniques which do not directly involve solving these mathematical problems. The full power of the mathematical machinery of Chapters 2 and 3 is felt here in the science of cryptology. Understanding the various aspects of cryptology hence becomes easier.

4.2 The Problems at a Glance

Let us first introduce the intractable problems of cryptology. In the rest of this chapter, we describe some known methods to solve these problems.

The *integer factorization problem* (*IFP*) is perhaps the most studied one in the lot. We know that $\mathbb{Z}$ is a unique factorization domain (UFD) (Definition 2.25, p40), that is, given a natural number n there are (pairwise distinct) primes $p_1, \ldots, p_r \in \mathbb{N}$ (unique up to rearrangement) such that $n = p_1^{\alpha_1} \cdots p_r^{\alpha_r}$ for some $\alpha_1, \ldots, \alpha_r \in \mathbb{N}$. Broadly speaking, the IFP is the determination of these p_i and α_i from the knowledge of n. Note that once the prime divisors p_i of n are known, it is rather easy to compute the multiplicities $\alpha_i = \mathrm{v}_{p_i}(n)$ by trial divisions. It is, therefore, sufficient to find out the primes p_i only. It is *easy* (Algorithm 3.13) to check if n is composite. If n is already prime, then its prime factorization is known. On the other hand, if n is known to be composite, an algorithm that splits n into two non-trivial factors, that is, that outputs $n_1, n_2 \in \mathbb{N}$ with $n = n_1 n_2$, $n_1 < n$ and $n_2 < n$, can be repeatedly used to compute the complete factorization of n. It is enough that a non-trivial factor n_1 of n is made available, the cofactor $n_2 = n/n_1$ is obtained by a single division. Finally, it is sometimes known a priori that n is the product of two (distinct odd) primes (as in the RSA protocols). In this case, the non-trivial split of n immediately gives the desired factorization of n. To sum up, the IFP can be stated in various versions, the presumed difficulty of all these versions being essentially the same.

Problem 4.1 **General integer factorization problem** Given an integer $n \in \mathbb{N}$, determine all the prime divisors of n.

Problem 4.2 **Integer factorization problem (IFP)** Given a composite integer $n \in \mathbb{N}$, find a non-trivial divisor n_1 of n (that is, a divisor n_1 of n in the range $1 < n_1 < n$).

Problem 4.3 **RSA integer factorization problem** Given a product $n = pq$ of two (distinct odd) primes p and q, find the prime divisors p and q of n.

Recall that if $n = p_1^{\alpha_1} \cdots p_r^{\alpha_r}$ is the prime factorization of n, then the Euler totient function $\phi(n)$ of n is $\phi(n) = \#\mathbb{Z}_n^* = n\left(1 - \frac{1}{p_1}\right) \cdots \left(1 - \frac{1}{p_r}\right)$. Thus, if the prime factorization of n is known, it is easy to compute $\phi(n)$. The converse is not known to be true in general. However, if $n = pq$ is the product of two primes, factoring n is polynomial-time equivalent to computing $\phi(n)$ (Exercise 3.6).

Problem 4.4 **Totient problem** Given a natural number $n \in \mathbb{N}$, compute $\phi(n)$.

Problem 4.5 **RSA totient problem** Given a product $n = pq$ of two (distinct odd) primes p and q, compute $\phi(n)$.

Note that $\mathbb{Z}[X]$ is also a UFD. Quite interestingly, it is computationally *easy* to find a non-trivial factor g of a polynomial $f \in \mathbb{Z}[X]$ (that is, $0 < \deg g < \deg f$). One might, for example, use the polynomial-time deterministic L^3 algorithm named after Lenstra, Lenstra and Lovasz (Section 4.8.2).

Square roots modulo an integer $n \in \mathbb{N}$ can be computed in probabilistic polynomial time, if n is a prime (Algorithm 3.16). If n is composite, the situation is different. If

the factorization of n is known, then the square roots can be computed modulo each prime divisor of n, lifted modulo the appropriate powers of the prime divisors and subsequently combined using the Chinese remainder theorem. On the other hand, if the factorization of n is not known, then computing square roots modulo n turns out to be a very difficult task. Recall that the Blum–Blum–Shub algorithm (Algorithm 3.37) exploits this fact to design a cryptographically secure random number generator.

Problem 4.6 **Modular square root problem (SQRTP)** Given a composite integer $n \in \mathbb{N}$ and an integer a, compute an integer x, if one exists, such that $x^2 \equiv a \pmod{n}$.

Let us now look at another class of problems of an apparently distinct flavour. Let G be a finite cyclic group of order $n := \#G$ and let g be a generator of G. For a moment, let us assume that G is multiplicatively written. Any element $a \in G$ can be written as $a = g^x$ for some integer x unique modulo n. In this case, x is called the *discrete logarithm* or the *index* of a with respect to the base g and is denoted by $\operatorname{ind}_g a$.

Problem 4.7 **Discrete logarithm problem (DLP)** Given a finite cyclic group G, a generator g of G and an element $a \in G$, compute $\operatorname{ind}_g a$.

If we now remove the restrictions that G is cyclic and/or that g is a generator of G (if G is cyclic), then we arrive at a generalized version of the DLP. Let us continue to assume that G is Abelian and finite. The subgroup H of G generated by $g \in G$ is anyway cyclic. If $a \in H$, then the discrete logarithm or index of a with respect to the base g is an integer x unique modulo $m := \operatorname{ord} H$ such that $a = g^x$. In this case, we denote such an integer x by $\operatorname{ind}_g a$. On the other hand, if $a \notin H$, then we say that the discrete logarithm $\operatorname{ind}_g a$ is *not defined*. Recall from Proposition 2.5 that if G is cyclic and if m is known, then checking if a belongs to H amounts to computing an exponentiation in G (that is, $a \in H$ if and only if a^m is the identity of G). If G is not cyclic (or if m is not known), then it is not easy, in general, to develop such a nice criterion.

Problem 4.8 **Generalized discrete logarithm problem (GDLP)** Given a finite Abelian group G and elements $g, a \in G$, determine if a belongs to the subgroup of G generated by g, and if so, compute $\operatorname{ind}_g a$.

Note that the DLP (or the GDLP) need not be an inherently difficult problem. Its difficulty depends on the choice of the group G and also on the representation of elements of G. For example, if G is the *additive* (cyclic) group $(\mathbb{Z}_n, +)$ and g is an integer with $\gcd(g, n) = 1$, then for every integer a we have $\operatorname{ind}_g a \equiv g^{-1}a \pmod{n}$, where the modular inverse $g^{-1} \pmod{n}$ can be computed efficiently using the extended gcd algorithm (Algorithm 3.8) on g and n. Also note that if G is cyclic and if each element of G is represented as $\operatorname{ind}_g a$ for a given generator g of G (see, for example, Section 2.9.3), then computing discrete logarithms in G to the base g is a trivial problem. In that case, it is also trivial to compute discrete logarithms (if existent) to any other base h (Exercise 4.3).

On the other hand, there are certain groups G in which discrete logarithms cannot be computed so easily; that is, computing indices in G may demand time not bounded by any polynomial in $\log n$, where $n = \operatorname{ord} G$. However, if the group operation on any two elements of G can be performed in time bounded by a polynomial in $\log n$, then cryptographic protocols can be based on G. Typical candidates for such groups are listed below together with the conventional names for the DLP over such groups.

Table 4.1 The discrete logarithm problem in various groups

Group	Name for the DLP
The (cyclic) multiplicative group $\mathbb{F}_q^*$ of a finite field $\mathbb{F}_q$	The *finite field discrete logarithm problem* or simply the DLP by an abuse of notation.
The (not necessarily cyclic) additive group $E(\mathbb{F}_q)$ of points of an elliptic curve defined over a finite field $\mathbb{F}_q$	The *Elliptic curve discrete logarithm problem* or the ECDLP
The Jacobian $\mathbb{J}_{\mathbb{F}_q}(C)$ of a hyperelliptic curve C defined over a finite field $\mathbb{F}_q$	The *Hyperelliptic curve discrete logarithm problem* or the HECDLP

Note that if we are interested in computing indices to a base $g \in G$, we may indeed replace, at least theoretically, G by the subgroup H of G generated by g and may assume, without loss of generality, that G is cyclic. Now, if we know an isomorphism $\varphi : G \to (\mathbb{Z}_n, +)$, computing discrete logarithms in G is rather easy (Exercise 4.4). However, computing such an isomorphism is, in general, not an easy task and may demand exponential time and/or storage requirements.

Another problem that is widely believed to be computationally equivalent to the DLP (at least for the groups mentioned in the above table) is called the *Diffie–Hellman problem* (DHP). Similar to the DLP, the DHP is presumably difficult to solve for the groups $\mathbb{F}_q^*$, $E(\mathbb{F}_q)$ and $\mathbb{J}_{\mathbb{F}_q}(C)$ and one may introduce the specific names DHP, ECDHP and HECDHP to designate this problem applied to these specific groups.

Problem 4.9 **Diffie–Hellman problem (DHP)** Let G be a multiplicative group and let $g \in G$. Given g^x and g^y for some (unknown) integers x and y, compute g^{xy}.

Clearly, if a solution of the DLP is given, one may compute $y = \mathrm{ind}_g(g^y)$ and, subsequently, $g^{xy} = (g^x)^y$. That is, the DHP is no harder than the DLP. A proof for the validity or otherwise of the converse relation between these two problems is not known. It is also widely believed that the DLP is computationally equivalent to the IFP. A complete proof of this equivalence is not known, though certain partial results are available in the literature.

There are some other difficult problems on which cryptographic systems can be built. Problem 4.10 deserves specific mention in this regard.

Problem 4.10 **Subset sum problem (SSP)** Given a set $A := \{a_1, \ldots, a_n\}$ of natural numbers and $s \in \mathbb{N}$, find out if there exist $\epsilon_1, \ldots, \epsilon_n \in \{0, 1\}$, such that $s = \sum_{i=1}^{n} \epsilon_i a_i$, that is, if there is a subset B of A with the property that $\sum_{a \in B} a = s$. The integers $a_1, \ldots, a_n$ are called the *weights* for the SSP.

The *Knapsack problem* is a related combinatorial optimization problem. In view of this, the set $\{a_1, \ldots, a_n\}$ is often called a *knapsack set*, and the SSP is, by an abuse of notation, also referred to as the knapsack problem.

Some of the early cryptographic systems based on the SSP have succumbed to efficient (even polynomial-time) cryptanalytic attacks. However, some schemes have been proposed in the recent years, which seem to be resistant to such attacks, or, in other words, for which good attacks are not yet known. As a result, it is important to study the SSP in some detail.

The SSP is often mapped to problems on lattices. Let $\mathbf{v}_1, \ldots, \mathbf{v}_n$ be linearly independent vectors in $\mathbb{R}^n$. Consider the set of integer linear combinations of these vectors:

$$L := \{a_1\mathbf{v}_1 + \cdots + a_n\mathbf{v}_n \mid a_1, \ldots, a_n \in \mathbb{Z}\}.$$

L is called the *lattice* generated by $\mathbf{v}_1, \ldots, \mathbf{v}_n$.

Problem 4.11 **Shortest vector problem (SVP)** Find a non-zero vector $\mathbf{v} \in L$ whose length $||\mathbf{v}||$ is smallest in L.

Problem 4.12 **Closest vector problem (CVP)** Given a vector $\mathbf{w} \in \mathbb{R}^n$, find a vector $\mathbf{v} \in L$ such that the length $||\mathbf{v} - \mathbf{w}||$ is smallest over all choices of $\mathbf{v} \in L$.

For some other difficult computational problems and their applications to cryptography, we refer the reader to the references suggested at the end of this chapter and of Chapter 5.

EXERCISE SET 4.2

4.1 **(a)** Let $n \geqslant 2$ be a square-free integer (that is, a product of pairwise distinct primes) and let $a \in \mathbb{N}$. Show that the exponentiation map $\varphi : \mathbb{Z}_n \to \mathbb{Z}_n$, $x \mapsto x^a$, is bijective if and only if $\gcd(a, \phi(n)) = 1$. **[H]**
(b) Show that if $n \in \mathbb{N}$ is not square-free, then for *no* integer $a \geqslant 2$ the exponentiation map $\mathbb{Z}_n \to \mathbb{Z}_n$, $x \mapsto x^a$, is bijective. **[H]**

4.2 Show that the following problems are polynomial-time reducible to the IFP.
(a) RSA key inversion problem (RSAKIP) Let $n = pq$ be a product of two (distinct odd) primes p and q. Given $e \in \mathbb{N}$ with $\gcd(e, \phi(n)) = 1$, compute an integer $d \in \mathbb{N}$ such that $ed \equiv 1 \pmod{\phi(n)}$.
(b) RSA problem (RSAP) Let n and e be as in Part (a). Given $c \in \mathbb{Z}_n$, compute $x \in \mathbb{Z}_n$ such that $c \equiv x^e \pmod{n}$. (By Exercise 4.1, such an x exists and is unique.)
(c) Quadratic residuosity problem (QRP) Given an odd integer $n > 1$ and an integer a with $\gcd(a, n) = 1$, check if a is a quadratic residue modulo n. (Note that if n is a prime, then this problem reduces to the computation of the Legendre symbol $\left(\frac{a}{n}\right)$. If, on the other hand, n is composite and $\left(\frac{a}{n}\right) = 1$, one cannot conclude that a is a quadratic residue modulo n.)

4.3 Let G be a finite cyclic group of order n and let g, g' be two arbitrary generators of G.
(a) Show that $\operatorname{ind}_g g'$ is invertible modulo n and that for every $a \in G$ we have $\operatorname{ind}_{g'} a \equiv (\operatorname{ind}_g a)(\operatorname{ind}_g g')^{-1} \pmod{n}$.
(b) Let $h \in G$, $m := \operatorname{ord}(h)$ and $y := \operatorname{ind}_g h$. Show that $m = n/\gcd(y, n)$, that $y/\gcd(y, n)$ is invertible modulo m and that for an arbitrary element $a \in G$ the index $\operatorname{ind}_h a$ exists if and only if $\gcd(y, n) | \operatorname{ind}_g a$ and in that case we have

$$\operatorname{ind}_h a \equiv (\operatorname{ind}_g a/\gcd(y, n))(y/\gcd(y, n))^{-1} \pmod{m}.$$

4.4 Let G be a finite cyclic multiplicatively written group of order n. An algorithm on G is said to be polynomial-time if it runs in time bounded above by a polynomial function

of $\log n$. Assume that the product of any two elements in G can be computed in polynomial time. Recall from Exercise 2.47 that $G \cong (\mathbb{Z}_n, +)$. Show that the computation of an isomorphism $\varphi : G \to \mathbb{Z}_n$ is polynomial-time equivalent to computing discrete logarithms in G. (That is, assuming that we are given a (two-way) black box that returns in polynomial time $\varphi(a)$ or $\varphi^{-1}(b)$ for every $a \in G$ and $b \in \mathbb{Z}_n$, discrete logarithms in G can be computed in polynomial time. Conversely, if discrete logarithms with respect to a primitive element $g \in G$ can be computed in polynomial time, then such a black box can be realized.)

4.5 Let p be an (odd) prime and let g be a primitive root modulo p. Show that $a \in \mathbb{Z}_p^*$ is a quadratic residue modulo p if and only if the index $\operatorname{ind}_g a$ is even. Hence, conclude that there is a polynomial-time (in $\log p$) algorithm that computes the least significant bit of $\operatorname{ind}_g a$, given any $a \in \mathbb{Z}_p^*$. More generally, let $p - 1 = 2^r s$, where $r, s \in \mathbb{N}$ and s is odd. Show that there exists a polynomial-time algorithm that computes the r least significant bits of $\operatorname{ind}_g a$ given any $a \in \mathbb{Z}_p^*$. (This exercise shows that the DLP has a polynomial-time solution for Fermat primes $F_n := 2^{2^n} + 1$. Note that F_n is prime for $n = 0, 1, 2, 3, 4$. No other Fermat primes are known.)

4.3 The Integer Factorization Problem

The *integer factorization problem* (*IFP*) (Problems 4.1, 4.2 and 4.3) is one of the most easily stated and yet hopelessly difficult computational problem that has attracted researchers' attention for ages and most notably in the age of electronic computers. A huge number of algorithms varying widely in the basic strategy, mathematical sophistication and implementation intricacy have been suggested, and, in spite of these, factoring a general integer having only 1000 bits seems to be an impossible task today even using the fastest computers on earth.

It is important to note here that even proving rigorous bounds on the running times of the integer-factoring algorithms is quite often a very difficult task. In many cases, we have to be satisfied with clever *heuristic* bounds based on one or more reasonable but unprovable assumptions.

This section highlights human achievements in the battle against the IFP. Before going into the details of this account we want to mention some relevant points. Throughout this section we assume that we want to factor a (positive) integer n. Since such an integer can be represented by $\lceil \lg(n+1) \rceil$ bits, the input size is taken to be $\lg n$ (or, $\ln n$, or $\log n$). Most modern factorization algorithms take time given by the following *subexponential* expression in $\ln n$:

$$L(n, \alpha, c) := \exp((c + \mathrm{o}(1))(\ln n)^{\alpha}(\ln \ln n)^{1-\alpha}),$$

where $0 < \alpha < 1$ and $c > 0$ are constants. As described in Section 3.2, the smaller the value of α is, the closer the expression $L(n, \alpha, c)$ is to a polynomial expression (in $\ln n$). If n is understood from the context, we write $L(\alpha, c)$ in place of $L(n, \alpha, c)$. Although the current best-known algorithms correspond to $\alpha = 1/3$, the algorithms with $\alpha = 1/2$ are also quite interesting. In this case, we use the shorter notation $L[c] := L(1/2, c)$.

Henceforth we will use, without explicit mention, the notation $q_1 := 2$, $q_2 := 3$, $q_3 := 5$, ... to denote the sequence of primes. The concept of q_t-smoothness (for some $t \in \mathbb{N}$) will often be referred to as B-smoothness, where $B = \{q_1, \ldots, q_t\}$. Recall from Theorem 2.21 that smaller integers have higher probability of being B-smooth for a given B. This observation plays an important role in designing integer factoring algorithms. The following special case of Theorem 2.21 is often useful.

Corollary 4.1 Let $n \in \mathbb{N}$, $x = \mathrm{O}(n^\alpha)$ and $y = L[\beta] = L(n, 1/2, \beta)$. Then we have the asymptotic formula $\psi(x, y) = L[-\frac{\alpha}{2\beta}]$.

Before any attempt of factoring n is made, it is worthwhile to check for the primality of n. Since probabilistic primality tests (like Algorithm 3.13) are quite efficient, we should first run one such test before we are sure that n is really composite. Henceforth, we will assume that n is *known* to be composite.

4.3.1 Older Algorithms

"Factoring in the dark ages" (a phrase attributed to Hendrik Lenstra) used fully exponential algorithms some of which are discussed now. Though the worst-case performances of these algorithms are quite poor, there are many situations when they might factor even a large integer quite fast. It is, therefore, worthwhile to spend some time on these algorithms.

Trial division

A composite integer n admits a factor $\leqslant \sqrt{n}$, that can be found by *trial divisions* of n by integers $\leqslant \sqrt{n}$. This demands $\mathrm{O}(\sqrt{n})$ trial divisions and is clearly impractical, even when n contains only 30 decimal digits. It is also true that n has a *prime* divisor $\leqslant \sqrt{n}$. So it suffices to carry out trial divisions by primes only. Though this modified strategy saves us many unnecessary divisions, the asymptotic complexity does not reduce much, since by the prime number theorem the number of primes $\leqslant \sqrt{n}$ is about $2\sqrt{n}/\ln n$. In addition, we need to have a list of primes $\leqslant \sqrt{n}$ or generate the primes on the fly, neither of which is really practical. A trade-off can be made by noting that an integer $m \geqslant 30$ cannot be prime unless $m \equiv 1, 7, 11, 13, 17, 19, 23, 29 \pmod{30}$. This means that we need to perform the trial divisions only by those integers m congruent to one of these values modulo 30 and this reduces the number of trial divisions to about 25 per cent. Though trial division is not a practical general-purpose algorithm for factoring large integers, we recommend extracting all the small prime factors of n, if any, by dividing n by a predetermined set $\{q_1, \ldots, q_t\}$ of small primes. If n is indeed q_t-smooth or has all prime factors $\leqslant q_t$ except only one, then the trial division method completely factors n quite fast. Even when n is not of this type, trial division might reduce its size, so that other algorithms run somewhat more efficiently.

Pollard's rho method

Pollard's rho method solves the IFP in an expected $\tilde{\mathrm{O}}(n^{1/4})$ time and is based on the birthday paradox (Exercise 2.172).

Let $p \leqslant \sqrt{n}$ be an (unknown) prime divisor of n and let $f : \mathbb{Z}_n \to \mathbb{Z}_n$ be a random map. We start with an initial value $x_1 \in \mathbb{Z}_n$ and generate a sequence $x_{i+1} = f(x_i)$, $i \in \mathbb{N}$, of elements of $\mathbb{Z}_n$. Let y_i denote the smallest non-negative integer satisfying $y_i \equiv x_i \pmod{p}$. By the birthday paradox, after $t = \mathrm{O}(\sqrt{p})$ iterates $x_1, \ldots, x_t$ are generated, we have a high chance that $y_i = y_j$, that is, $x_i \equiv x_j \pmod{p}$ for some $1 \leqslant i < j \leqslant t$. This means that $p|(x_i - x_j)$ and computing $\gcd(x_i - x_j, n)$ splits n into two non-trivial factors with high probability. The method fails if this gcd is n. For a random n, this incident of having a gcd equal to n is of very low probability.

Algorithm 4.1 gives a specific implementation of this method. Computing gcds for all the pairs $(x_i - x_j, n)$ is a massive investment of time. Instead we store (in the variable ξ) the values x_r, $r = 2^t$, for $t \in \mathbb{N}$ and compute only $\gcd(x_{r+s} - x_r, n)$ for $s = 1, \ldots, r$. Since the sequence y_i, $i \in \mathbb{N}$, is ultimately periodic with expected length of period $\tau = \mathrm{O}(\sqrt{p})$, we eventually reach a t with $r = 2^t \geqslant \tau$. In that case, the for loop detects a match. Typically, the update function f is taken to be $f(x) = x^2 - 1 \pmod{n}$, which, though not a random function, *behaves* like one. Note that the iterates y_i, $i \in \mathbb{N}$, may be visualized as being located on the Greek letter ρ as shown in Figure 4.1 (with a tail of the first μ iterates followed by a cycle of length τ). This is how this method derives its name.

Algorithm 4.1 takes an expected running time $\tilde{\mathrm{O}}(\sqrt{p})$. Since $p = \mathrm{O}(\sqrt{n})$, Pollard's rho method runs in expected time $\tilde{\mathrm{O}}(\sqrt[4]{n})$.

Algorithm 4.1 **Pollard's rho method**

Input: A composite integer $n \in \mathbb{N}$.
Output: A non-trivial factor of n.
Steps:
Choose a random element $x \in \mathbb{Z}_n$ and set $\xi := x$ and $r := 1$.
while (1) {
  for $s = 1, \ldots, r$ {
    $x := f(x)$.
    $d := \gcd(x - \xi, n)$.
    if $(1 < d < n)$ { Return d. }
  }
  $\xi := x$.
  $r := 2r$.
}

Many modifications of Pollard's rho method have been proposed in the literature. Perhaps the most notable one is an idea due to R. P. Brent. All these modifications considerably speed up Algorithm 4.1, though leaving the complexity essentially the same, that is, $\tilde{\mathrm{O}}(n^{1/4})$. We will not describe these modifications in this book.

Pollard's $p - 1$ method

Pollard's $p - 1$ method is dependent on the prime factors of $p - 1$ for a prime divisor p of n. Indeed if $p - 1$ is rather smooth, this method may extract a (non-trivial) factor of

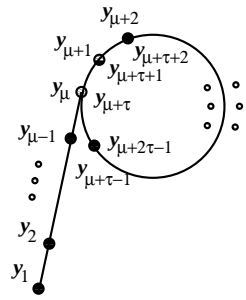


Figure 4.1 Iterates in Pollard's rho method

n pretty fast, even when p itself is quite large. To start with we extend the definition of smoothness as follows.

Definition 4.1 Let $y \in \mathbb{N}$. An integer x is called *y-power-smooth* if, whenever a prime power p^e divides x, we have $p^e \leqslant y$. Clearly, a y-power-smooth integer is y-smooth, but not necessarily conversely. □

Let p be an (unknown) prime divisor of n. We may assume, without loss of generality, that $p \leqslant \sqrt{n}$. Assume that $p-1$ is M-power-smooth. Then $(p-1) | \operatorname{lcm}(1, \ldots, M)$ and, therefore, for an integer a with $\gcd(a, n) = 1$ (and hence with $\gcd(a, p) = 1$), we have $a^{\operatorname{lcm}(1,\ldots,M)} \equiv 1 \pmod{p}$ by Fermat's little theorem, that is, $d := \gcd(a^{\operatorname{lcm}(1,\ldots,M)} - 1, n) > 1$. If $d \neq n$, then d is a non-trivial factor of n. In case we have $d = n$ (a very rare occurrence), we may try with another a or declare failure.

The problem with this method is that p and so M are not known in advance. One may proceed by guessing successively increasing values of M, till the method succeeds. In the worst case, that is, when p is a safe prime, we have $M = (p-1)/2$. Since $p \leqslant \sqrt{n}$, this algorithm runs in a worst-case time of $\tilde{\mathrm{O}}(\sqrt{n})$. However, if M is quite small, then this algorithm is rather efficient, irrespective of how large p itself is.

In Algorithm 4.2, we give a variant of the $p-1$ method, where we supply a predetermined value of the bound M. We also assume that we have at our disposal a precalculated list of all primes $q_1, \ldots, q_t \leqslant M$.

There is a modification of this algorithm known as *Stage 2* or the *second stage*. For this, we choose a second bound M' larger than M. Assume that $p - 1 = rq$, where r is M-power-smooth and q is a prime in the range $M < q \leqslant M'$. In this case, Stage 2 computes with high probability a factor of n after doing an $\tilde{\mathrm{O}}(\sqrt{M'})$ operations as follows. When Algorithm 4.2 returns "failure" at the last step, it has already computed the value $A := a^m \pmod{n}$, where $m = \prod_{i=1}^{t} q_i^{e_i}$, $e_i = \lfloor \ln M / \ln q_i \rfloor$. In this case, A has the multiplicative order of q modulo p, that is, the subgroup H of $\mathbb{Z}_p^*$ generated by A has order q. We choose $s = \mathrm{O}(\sqrt{q})$ random integers $l_1, \ldots, l_s \in \{0, \ldots, q-1\}$. By the birthday paradox (Exercise 2.172), we have with high probability $A^{l_i} \equiv A^{l_j} \pmod{p}$ for some $i \neq j$. In that case, $d := \gcd(A^{l_i} - A^{l_j}, n)$ is divisible by p and is a desired factor of n (unless $d = n$, a case that occurs with a very low probability). In practice, we do not know q and so we determine s and the integers $l_1, \ldots, l_s$ using the bound M' instead of q.

Algorithm 4.2 **Pollard's $p-1$ method**

Input: A composite integer $n \in \mathbb{N}$, a bound M and all primes $q_1, \ldots, q_t \leqslant M$.
Output: A non-trivial factor d of n or "failure".
Steps:
Select a random integer a, $1 < a < n$. /* For example, we may take $a := 2$ */
if $(d := \gcd(a, n) \neq 1)$ { Return d. }
for $i = 1, \ldots, t$ {
  $e_i := \lfloor \ln M / \ln q_i \rfloor$.
  $a := a^{q_i^{e_i}} \pmod{n}$.
  $d := \gcd(a - 1, n)$.
  if $(1 < d < n)$ { Return d. }
  if $(d = n)$ { Return "failure". } /* Or repeat the for loop with another a */
  if $(d = 1)$ { Return "failure". }
}
Return "failure".

In another variant of Stage 2, we compute the powers $A^{q_{t+1}}, \ldots, A^{q_{t'}} \pmod{n}$, where $q_{t+1}, \ldots, q_{t'}$ are all the primes q_j satisfying $M < q_j \leqslant M'$. If $p - 1 = rq$ is of the desired form, we would find $q = q_j$ for some $t < j \leqslant t'$, and then $\gcd(A^q - 1, n)$, if not equal to n, would be a non-trivial factor of n.

In practice, one may try one's luck using this algorithm for some M in the range $10^5 \leqslant M \leqslant 10^6$ (and possibly also the second stage with $10^6 \leqslant M' \leqslant 10^8$) before attempting a more sophisticated algorithm like the MPQSM, the ECM or the NFSM.

Williams' $p+1$ method

As always, we assume that n is a composite integer and that p is an (unknown) prime divisor of n. Pollard's $p-1$ method uses an element a in the group $\mathbb{F}_p^*$ whose multiplicative order is $p-1$. The idea of Williams' $p+1$ method is very similar, that is, it works with an element a, this time in $\mathbb{F}_{p^2}^*$, whose multiplicative order is $p+1$. If $p+1$ is M-power-smooth for a reasonably small bound M, then computing $d := \gcd(a^{p+1} - 1, n) > 1$ splits n with high probability.

In order to find an element $a \in \mathbb{F}_{p^2}^*$ of order $p+1$, we proceed as follows. Let α be an integer such that $\alpha^2 - 4$ is a quadratic non-residue modulo p. Then the polynomial $f(X) := X^2 - \alpha X + 1 \in \mathbb{F}_p[X]$ is irreducible and $\mathbb{F}_{p^2} \cong \mathbb{F}_p[X]/\langle f \rangle$. Let $a, b \in \mathbb{F}_{p^2} \setminus \mathbb{F}_p$ be the two roots of f. Then $ab = 1$ and $a + b = \alpha$. Since $f(a^p) = 0$ (check it!) and since $a \notin \mathbb{F}_p$, we have $a^p = b = a^{-1}$, that is, $a^{p+1} = 1$.

Unfortunately, p is not known in advance. Therefore, we represent elements of $\mathbb{F}_p$ as integers modulo n and the elements of $\mathbb{F}_{p^2}$ as polynomials $c_0 + c_1 X$ with $c_0, c_1 \in \mathbb{Z}_n$. Multiplying two such elements of $\mathbb{F}_{p^2}$ is accomplished by multiplying the two polynomials representing these elements modulo the defining polynomial $f(X)$, the coefficient arithmetic being that of $\mathbb{Z}_n$. This gives us a way to do exponentiations in $\mathbb{F}_{p^2}$ in order to compute $a^m - 1$ for a suitable m (for example, $m = \operatorname{lcm}(1, \ldots, M)$).

However, the absence of knowledge of p has a graver consequence, namely, it is impossible to decide whether $\alpha^2 - 4$ is a quadratic non-residue modulo p for a given

integer α. The only thing we can do is to try several random values of α. This is justified, because if k random integers α are tried, then the probability that for all of these α the integers $\alpha^2 - 4$ are quadratic residues modulo p is only $1/2^k$.

The code for the $p+1$ method is very similar to Algorithm 4.2. We urge the reader to complete the details. Since $p^3 - 1 = (p-1)(p^2+p+1)$, $p^4 - 1 = (p^2-1)(p^2+1)$ and so on, we can work in higher extensions like $\mathbb{F}_{p^3}$, $\mathbb{F}_{p^4}$ to find elements of order p^2+p+1, p^2+1 and so on, and generalize the $p \pm 1$ methods. However, the integers p^2+p+1, p^2+1, being large (compared to $p \pm 1$), have smaller chance of being M-smooth (or M-power-smooth) for a given bound M.

The reader should have recognized why we paid attention to strong primes and safe primes (Definition 3.5, p 199, and Algorithm 3.14, p 200). Let us now concentrate on the recent developments in the IFP arena.

4.3.2 The Quadratic Sieve Method

Carl Pomerance's *quadratic sieve method* (*QSM*) is one of the (reasonably) successful modern methods of factoring integers. Though the number field sieve factoring method is the current champion, there was a time in the recent past when the quadratic sieve method and the elliptic curve method were known to be the fastest algorithms for solving the IFP.

The basic algorithm

We assume that n is a composite integer which is not a perfect square (because it is easy to detect if n is a perfect square and if so, we replace n by $\sqrt{n}$). The basic idea is to reach at a congruence of the form

$$x^2 \equiv y^2 \pmod{n} \tag{4.1}$$

with $x \not\equiv \pm y \pmod{n}$. In that case, $\gcd(x-y, n)$ is a non-trivial factor of n.

We start with a *factor base* $B = \{q_1, \ldots, q_t\}$ comprising the first t primes and let $H := \lceil \sqrt{n} \rceil = \lfloor \sqrt{n} \rfloor + 1$ and $J := H^2 - n$. Then H and J are each $\mathrm{O}(\sqrt{n})$ and hence for a small integer c the right side of the congruence

$$(H+c)^2 \equiv J + 2cH + c^2 \pmod{n}$$

is also $\mathrm{O}(\sqrt{n})$. We try to factor $T(c) := J + 2cH + c^2$ using trial divisions by elements of B. If the factorization is successful, that is, if $T(c)$ is B-smooth, then we get a *relation* of the form

$$T(c) = q_1^{\alpha_1} \cdots q_t^{\alpha_t} \equiv (H+c)^2 \pmod{n}, \tag{4.2}$$

where $\alpha_1, \ldots, \alpha_t \in \{0, 1, 2, \ldots\}$. (Note that $T(c) \neq 0$, since n is assumed *not* to be a perfect square.) If all α_i are even, say, $\alpha_i = 2\beta_i$, then we get the desired Congruence (4.1) with $x = q_1^{\beta_1} \cdots q_t^{\beta_t}$ and $y = H + c$. But this is rarely the case. So we keep on generating other relations. After sufficiently many relations are available, we combine these together (by multiplication) to get Congruence (4.1) and compute $\gcd(x-y, n)$.

If this does not give a non-trivial factor, we try to recombine the collected relations in order to get another Congruence (4.1). This is how Pomerance's QSM works.

In order to find suitable combinations for yielding Congruence (4.1), we employ a method similar to Gaussian elimination. Assume that we have collected r relations of the form

$$(H + c_i)^2 \equiv \prod_{j=1}^{t} q_j^{\alpha_{ij}} \pmod{n},\ i = 1, \ldots, r\,.$$

We search for integers $\beta_1, \ldots, \beta_r \in \{0, 1\}$ such that the product

$$\prod_{i=1}^{r} (H + c_i)^{2\beta_i} = \left(\prod_{i=1}^{r} (H + c_i)^{\beta_i} \right)^2 \equiv \prod_{j=1}^{t} q_j^{\sum_{i=1}^{r} \beta_i \alpha_{ij}} \pmod{n}$$

is a desired Congruence (4.1). The left side of this congruence is already a square. In order to make the right side a square too, we have to essentially solve the following system of linear congruences modulo 2:

$$\sum_{i=1}^{r} \beta_i \alpha_{ij} \equiv 0 \pmod{2},\ j = 1, \ldots, t\,.$$

This is a system of t equations over $\mathbb{F}_2$ in r unknowns $\beta_1, \ldots, \beta_r$ and is expected to have solutions, if r is slightly larger than t. Note that the values of α_{ij} modulo 2 are only needed for solving the above linear system. This means that we can have a compact representation of the coefficient matrix (α_{ij}) by packing 32 of the coefficients as bits per word. Gaussian elimination (over $\mathbb{F}_2$) can be done using bit operations only.

The running time of this method can be derived using Corollary 4.1. Note that the integers $T(c)$ that are tested for B-smoothness are $\mathrm{O}(n^{1/2})$ which corresponds to $\alpha = 1/2$ in the corollary. We take $q_t = L[1/2]$ (so that $t = L[1/2]/\ln L[1/2] = L[1/2]$ by the prime number theorem) which corresponds to $\beta = 1/2$. Assuming that the integers $T(c)$ behave as random integers of magnitude $\mathrm{O}(n^{1/2})$, the probability that one such $T(c)$ is B-smooth is $L[-1/2]$. Therefore, if $L[1]$ values of c are tried, we expect to get $L[1/2]$ relations involving the $L[1/2]$ primes $q_1, \ldots, q_t$. Combining these relations by Gaussian elimination is now expected to produce a non-trivial Congruence (4.1). This gives us a running-time of the order of $L[3/2]$ for the relation collection stage. Gaussian elimination using $L[1/2]$ unknowns also takes asymptotically the same time. However, each $T(c)$ can have at most $\mathrm{O}(\log n)$ distinct prime factors, implying that Relation (4.2) is necessarily sparse. This sparsity can be effectively exploited and the Gaussian elimination can be done essentially in time $L[1]$. Nevertheless, the entire procedure runs in time $L[3/2]$, a subexponential expression in $\ln n$.

Sieving

In order to reduce the running time from $L[3/2]$ to $L[1]$, we employ what is known as *sieving* (and from which the algorithm derives its name). Let us fix a priori the *sieving interval*, that is, the values of c for which $T(c)$ is tested for B-smoothness, to be $-M \leqslant c \leqslant M$, where $M = L[1]$. Let $q \in B$ be a small prime (that is, $q = q_i$

for some $i = 1, \ldots, t$). We intend to find out the values of c such that $q^h | T(c)$ for small exponents $h = 1, 2, \ldots$. Since $T(c) = J + 2cH + c^2 = (c + H)^2 - n$, the solvability for c of the condition $q^h | T(c)$ or of $q | T(c)$ is equivalent to the solvability of the congruence $(c + H)^2 \equiv n \pmod{q}$. If n is a quadratic non-residue modulo q, no c satisfies the above condition. Consequently, the factor base B may comprise only those primes q for which n is a quadratic residue modulo q (instead of all primes $\leqslant q_t$). So we assume that q meets this condition. We may also assume that $q \not| n$, because it is a good strategy to perform trial divisions of n by all the primes in B before we go for sieving. The sieving process makes use of an array $\mathfrak{A}$ indexed by c. We initialize the array location $\mathfrak{A}_c := \ln |T(c)|$ for each c, $-M \leqslant c \leqslant M$.

We explain the sieving process only for an odd prime q. The modifications for the case $q = 2$ are left to the reader as an easy exercise. The congruence $x^2 - n \equiv 0 \pmod{q}$ has two distinct solutions for x, say, x_1 and x_1' mod q. These correspond to two solutions for c of $(H + c)^2 \equiv n \pmod{q}$, namely, $c_1 \equiv x_1 - H \pmod{q}$ and $c_1' \equiv x_1' - H \pmod{q}$. For each value of c in the interval $-M \leqslant c \leqslant M$, that is congruent either to c_1 or c_1' modulo q, we subtract $\ln q$ from $\mathfrak{A}_c$. We then lift the solutions x_1 and x_1' to the (unique) solutions x_2 and x_2' of the congruence $x^2 - n \equiv 0 \pmod{q^2}$ (Exercise 3.29), compute $c_2 \equiv x_2 - H \pmod{q^2}$ and $c_2' \equiv x_2' - H \pmod{q^2}$ and for each c in the range $-M \leqslant c \leqslant M$ congruent to c_2 or c_2' modulo q^2 subtract $\ln q$ from $\mathfrak{A}_c$. We then again lift to obtain the solutions modulo q^3 and proceed as above. We repeat this process of lifting and subtracting $\ln q$ from appropriate locations of $\mathfrak{A}$ until we reach a sufficiently large $h \in \mathbb{N}$ for which neither c_h nor c_h' corresponds to any value of c in the range $-M \leqslant c \leqslant M$. We then choose another q from the factor base and repeat the procedure explained in this paragraph for this q.

After the sieving procedure is carried out for all small primes q in the factor base B, we check for which c, $-M \leqslant c \leqslant M$, the array location $\mathfrak{A}_c$ is 0. These are precisely the values of c in the indicated range for which $T(c)$ is B-smooth. For each smooth $T(c)$, we then compute Relation (4.2) using trial division (by primes of B).

The sieving process replaces trial divisions (of every $T(c)$ by every q) by subtractions (of $\ln q$ from appropriate $\mathfrak{A}_c$). This is intuitively the reason why sieving speeds up the relation collection stage. For a more rigorous analysis of the running time, note that in order to get the desired c_i and c_i' modulo q^i for each $q \in B$ and for each $i = 1, \ldots, h$ we have either to compute a square root modulo q (for $i = 1$) or to solve a congruence (during lifting for $i \geqslant 2$), each of which can be done in polynomial time. Also the bound h on the exponent of q satisfy $q^h = \mathrm{O}(\sqrt{n})$, that is, $h = \mathrm{O}(\log n)$. Finally, there are $L[1/2]$ primes in B. Therefore, the computation of the c_i and c_i' for all q and i takes a total of $L[1/2]$ time.

Now, we count the total number ν of subtractions of different $\ln q$ values from all the locations of the array $\mathfrak{A}$. The size of $\mathfrak{A}$ is $2M + 1$. For each q^i, we need to subtract $\ln q$ from at most $2\left\lceil (2M + 1)/q^i \right\rceil$ locations (for odd q), and we also have $q^i = \mathrm{O}(\sqrt{n})$. Therefore, ν is of the order of $2(2M + 1) \sum_{q,i} 1/q^i \leqslant 2(2M + 1) \sum_{r=1}^{Q} 1/r = 2(2M + 1) H_Q$, where Q is the maximum of all q^i and is $\mathrm{O}(\sqrt{n})$, and where H_m, $m \in \mathbb{N}$, denote the harmonic numbers (Exercise 4.6). But $H_m = \mathrm{O}(\ln m)$, and so $\nu = \mathrm{O}(2(2M + 1) \log n) = L[1]$, since $M = L[1]$.

The logarithms $\ln q$ (as well as the initial array values $\ln |T(c)|$) are irrational numbers and hence need infinite precision for storing. We, however, need to work with only crude approximations of these logarithms, say up to three places after the decimal point. In that case, we cannot take $\mathfrak{A}_c = 0$ as the criterion for selecting smooth values of $T(c)$, because the approximate representation of logarithms leads to truncation (and/or rounding) errors. In practice, this is not a severe problem, because $T(c)$ is *not* smooth if and only if it has a prime factor at least as large as q_{t+1} (the smallest prime not in B). This implies that at the end of the sieving operation the values of $\mathfrak{A}_c$ for smooth $T(c)$ are close to 0, whereas those for non-smooth $T(c)$ are much larger (close to a number at least as large as $\ln q_{t+1}$). Thus we may set the selection criterion for smooth integers as $|\mathfrak{A}_c| \leqslant 1$ or as $|\mathfrak{A}_c| \leqslant 0.1 \ln q_{t+1}$. It is also possible to replace floating point subtraction by integer subtraction by doing the arithmetic on 1000 times the logarithm values. To sum up, the $\nu = L[1]$ subtractions the sieving procedure does would be only single-precision operations and hence take a total of $L[1]$ time.

As mentioned earlier, Gaussian elimination with sparse equations can also be performed in time $L[1]$. So Pomerance's algorithm with sieving takes time $L[1]$.

Incomplete sieving

Numerous modifications over this basic strategy speed up the algorithm reasonably. One possibility is to do sieving every time only for $h = 1$ and ignore all higher powers of q. That is, for every q we check which of the integers $T(c)$ are divisible by q and then subtract $\ln q$ from the corresponding indices of the array $\mathfrak{A}$. If some $T(c)$ is divisible by a higher power of q, this strategy fails to subtract $\ln q$ the required number of times. As a result, this $T(c)$, even if smooth, may fail to pass the smoothness criterion. This problem can be overcome by increasing the cut-off from 1 (or $0.1 \ln q_{t+1}$) to a value $\xi \ln q_t$ for some $\xi \geqslant 1$. But then some non-smooth $T(c)$ will pass through the selection criterion in addition to some smooth ones that could not, otherwise, be detected. This is reasonable, because the non-smooth ones can be later filtered out from the smooth ones and one might use even trial divisions to do so. Experimentations show that values of $\xi \leqslant 2.5$ work quite well in practice.

The reason why this strategy performs well is as follows. If q is small, for example $q = 2$, we should subtract *only* 0.693 from $\mathfrak{A}_c$ for every power of 2 dividing $T(c)$. On the other hand, if q is much larger, say $q = 1{,}299{,}709$ (the 10^5-th prime), then $\ln q \approx 14.078$ is *large*. But $T(c)$ would not, in general, be divisible by a *high* power of this q. This modification, therefore, leads to a situation where the probability that a smooth $T(c)$ is actually detected as smooth is quite high. A few relations would still be missed out even with the modified selection criterion, but that is more than compensated by the speed-up gained by the method. Henceforth, we will call this modified strategy as *incomplete sieving* and the original strategy (of considering all powers of q) as *complete sieving*.

Large prime variation

Another trick known as *large prime variation* also tends to give more usable relations than are available from the original (complete or incomplete) sieving. In this context,

we call a prime q' *large*, if $q' \notin B$. A value of $T(c)$ is often expected to be B-smooth except for a single large prime factor:

$$T(c) = q_1^{\alpha_1} \cdots q_t^{\alpha_t} q' \tag{4.3}$$

with $q' \notin B$. Such a value of $T(c)$ can be easily detected. For example, incomplete sieving with the relaxed selection criterion is expected to give many such relations *naturally*, whereas for complete sieving, if the left-over of $\ln |T(c)|$ in $\mathfrak{A}_c$ at the end of the subtraction steps is $< 2 \ln q_t$, then this must correspond to a large prime factor $< q_t^2$. Instead of throwing away an apparently unusable Equation (4.3), we may keep track of them. If a large prime q' is not large enough (that is, not much larger than q_t), then it might appear on the right side of Equation (4.3) for more than one values of c, and if that is the case, all these relations taken together now become usable for the subsequent Gaussian elimination stage (after including q' in the factor base). This means that for each large prime occurring more frequently than once, the factor base size increases by 1, whereas the number of relations increases by at least 2. Thus with a little additional effort we enrich the factor base and the relations collected, and this, in turn, increases the probability of finding a useful Congruence (4.1), our ultimate goal. Viewed from another angle, the strategy of large prime variation allows us to start with smaller values of t and/or M and thereby speed up the sieving stage and still end up with a system capable of yielding the desired Congruence (4.1). Note that an increased factor base size leads to a larger system to solve by Gaussian elimination. But this is not a serious problem in practice, because the sieving stage (and not the Gaussian elimination stage) is usually the bottleneck of the running time of the algorithm.

It is natural that the above discussion on handling *one* large prime is applicable to situations where a $T(c)$ value has more than one large prime factors, say q' and q''. Such a $T(c)$ value leads to a usable relation if $q_t^2 < q'q'' < q_t^3$. This situation can be detected by a compositeness test on the non-smooth part of $T(c)$. Subsequently, we have to factor the non-smooth part to obtain the two large primes q' and q''. This is called *two large prime variation*. As the size of the integer n to be factored becomes larger, one may go for three and four large prime variations.

We will shortly encounter many other instances of sieving (for solving the IFP and the DLP). Both incomplete sieving and the use of large primes, if carefully applied, help speed up most of these sieving methods much in the same way as they do in connection with the QSM.

The multiple polynomial quadratic sieve

Easy computations (Exercise 4.11) show that the average and maximum of the integers $|T(c)|$ checked for smoothness in the QSM are approximately MH and $2MH$ respectively. Though these values are theoretically $\mathrm{O}(\sqrt{n})$, in practice the factor of M (or $2M$) makes the integers $|T(c)|$ somewhat large leading to a poor yield of B-smooth integers for larger values of $|c|$ in the sieving interval. The *multiple-polynomial quadratic sieve method* (MPQSM) applies a nice trick to reduce these average and maximum values. In the original QSM, we work with a single polynomial in c, namely,

$$T(c) = J + 2cH + c^2 = (H + c)^2 - n.$$

Now, we work with a more general quadratic polynomial

$$\bar{T}(c) = U + 2Vc + Wc^2 \in \mathbb{Z}[c]$$

with $W > 0$ and $V^2 - UW = n$. (The original $T(c)$ corresponds to $U = J$, $V = H$ and $W = 1$.) Then we have $W\bar{T}(c) = (Wc + V)^2 - (V^2 - UW)$, that is, in this case a relation looks like

$$W\bar{T}(c) \equiv (Wc + V)^2 \pmod{n}.$$

This relation has an additional factor of W that was absent in Relation (4.2). However, if W is chosen to be a prime (possibly a *large* one), then the Gaussian elimination stage proceeds exactly as in the original method. Indeed in this case W appears in every relation and hence poses no problem. Only the integers $\bar{T}(c)$ need to be checked for B-smoothness and hence should have *small* values. The sieving procedure (that is, computing the appropriate locations of $\mathfrak{A}$ for subtracting $\ln q$, $q \in B$) for the general polynomial $\bar{T}(c)$ is very much similar to that for $T(c)$. The details are left to the reader as an easy exercise.

Let us now explain how we can choose the parameters U, V, W. To start with we fix a suitable sieving interval $-\bar{M} \leqslant c \leqslant \bar{M}$ and then choose W to be a prime close to $\sqrt{2n}/\bar{M}$ such that n is a quadratic residue modulo W. Then we compute a square root V of n modulo W (Algorithm 3.16) and finally take $U = (V^2 - n)/W$. This choice clearly gives $W \approx \sqrt{2n}/\bar{M}$, $0 < V < W$ and $U \approx -\sqrt{n/2}\bar{M}$. (Indeed one may choose $0 < V < W/2$, but this is not an important issue.) Now, the maximum value of $|\bar{T}(c)|$ becomes $\bar{T}(\bar{M}) = U + 2V\bar{M} + W\bar{M}^2 \approx -\sqrt{n/2}\bar{M} + 2V\bar{M} + \sqrt{2n}\bar{M} \approx \sqrt{n/2}\bar{M} \approx (\bar{M}/\sqrt{2})H$. Thus even for $\bar{M} = M$, this maximum value is smaller by a factor of $2\sqrt{2} \approx 2.828$ than the maximum value of $|T(c)|$ in the original QSM. Moreover, we may choose somewhat smaller values of $\bar{M}$ (compared to M) by working with *several* polynomials $\bar{T}(c)$ corresponding to different choices for the prime W. This is why the MPQSM, despite having the same theoretical running-time ($L[1]$) as the original QSM, runs faster in practice.

Parallelization

The QSM is highly parallelizable. More specifically, different processors can handle pairwise disjoint subsets of B during the sieving process. That is, each processor P maintains a local array $\mathfrak{A}_P$ indexed by c, $-M \leqslant c \leqslant M$. The (local) sieving process at P starts with initializing all the locations $(\mathfrak{A}_P)_c$ to 0. For each prime q in the subset B_P of the factor base B assigned to P, one *adds* $\ln q$ to appropriate locations (and appropriate numbers of times). After all these processors finish local sieving, a central processor computes, for each c in the sieving interval, the value $\ln |T(c)| - \sum_P (\mathfrak{A}_P)_c$ (where the sum extends over all processors P which have done local sieving) based on which $T(c)$ is recognized as smooth or not. For the multiple-polynomial variant of the QSM, different processors might handle different polynomials $\bar{T}(c)$ and/or different subsets of B.

TWINKLE: Shamir's factoring device

Adi Shamir has proposed the complete design of a (hardware) device, *TWINKLE* (*The Weizmann INstitute Key Location Engine*), that can perform the sieving stage of QSM a hundred to thousand times faster than software implementations in usual PCs available nowadays. This speed-up is obtained by using a high clock speed (10 GHz) and opto-electronic technology for detecting smooth integers. Each TWINKLE, if mass produced, has an estimated cost of US $5,000.

The working of TWINKLE is described in Figure 4.2. It uses an opaque cylinder of a height of about 10 inches and a diameter of about 6 inches. At the bottom of the cylinder is an array of LEDs,[1] each LED representing a prime in the factor base. The i-th LED (corresponding to the i-th prime q_i) emits light of intensity proportional to $\log q_i$. The device is clocked and the i-th LED emits light only during the clock cycles c for which $q_i | T(c)$. The light emitted by all the active LEDs at a given clock cycle is focused by a lens and a photo-detector senses the total emitted light. If this total light exceeds a certain threshold, the corresponding clock cycle (that is, the time c) is reported to a PC attached to TWINKLE. The PC then analyses the particular $T(c)$ for smoothness over $\{q_1, \ldots, q_t\}$ by trial division.

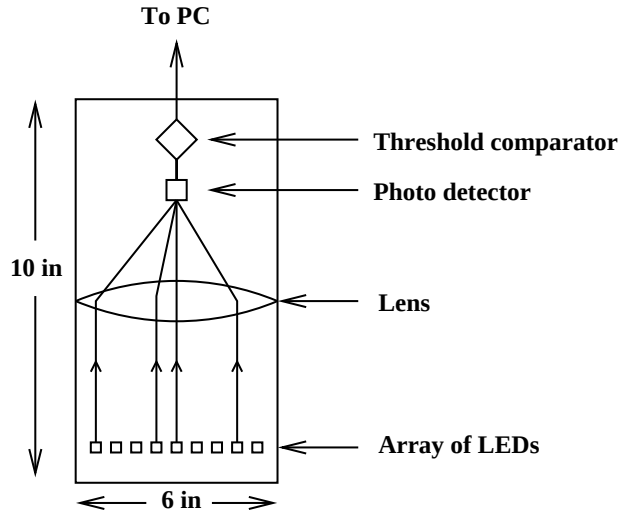


Figure 4.2 Working of TWINKLE

Thus, TWINKLE implements incomplete sieving by opto-electronic means. The major difference between TWINKLE's sieving and software sieving is that in the latter we used an array of *times* (the c values) and the iteration went over the set of small primes. In TWINKLE, we use an array of small primes and allow time to iterate over the different values of c in the sieving interval $-M \leqslant c \leqslant M$. An electronic circuit

[1] An LED (light emitting diode) is an electronic device that emits light, when current passes through it. A GaAs(Gallium arsenide)-based LED emits (infra-red) light of wavelength ∼870 nano-meters. In the operational range of an LED, the intensity of emitted light is roughly proportional to the current passing through the LED.

in TWINKLE computes for each LED the cycles c at which that LED is expected to emanate light. That is to say that the i-th LED emits light only in the clock cycles c congruent modulo q_i to any of the two solutions c_1 and c_1' of $T(c) \equiv 0 \pmod{q_i}$. Shamir's original design uses *two* LEDs for each prime q_i, one corresponding to c_1, the other to c_1'. In that case, each LED emits light at regularly spaced clock cycles and this simplifies the electronic circuitry (at the cost of having twice the number of LEDs).

Another difference of TWINKLE from software sieving is that here we *add* the $\log q_i$ values (to zero) instead of subtracting them from $\log |T(c)|$. By Exercise 4.11, the values $|T(c)|$ typically have variations by small constant factors. Taking logs reduces this variation further and, therefore, comparing the sum of the active $\log q_i$ values for a given c with a fixed predefined threshold (say $\log MH$) independent of c is a neat way of bypassing the computation of all $\log |T(c)|$, $-M \leqslant c \leqslant M$. (This strategy can also be used for software sieving.)

The reasons, why TWINKLE speeds up the sieving procedure over software implementations in conventional PCs, are the following:

(1) Silicon-based PC chips at present can withstand clock frequencies on the order of 1 GHz. On the contrary a GaAs-based wafer containing the LED array can be clocked faster than 10 GHz.

(2) There is no need to initialize the array $\mathfrak{A}$ (to $\log |T(c)|$ or zero). Similarly at the end, there is no need to compare the final values in all these array locations with a threshold.

(3) The addition of all the $\log q_i$ values effective at a given c is done instantly by analog optical means. We do not require an explicit electronic adder.

Shamir [269] reports the full details of a VLSI[2] design of TWINKLE.

* 4.3.3 Factorization Using Elliptic Curves

H. W. Lenstra's *elliptic curve method* (*ECM*) is another *modern* algorithm to solve the IFP and runs in expected time $L(p, 1/2, \sqrt{2})$, where p is the smallest prime factor of n (the integer to be factored). Since $p \leqslant \sqrt{n}$, this running time is $L[1] = L(n, 1/2, 1)$: that is, the same as the QSM. However, if p is small (that is, if $p = \mathrm{O}(n^\alpha)$ for some $\alpha < 1/2$), then the ECM is expected to outperform the QSM, since the working of the QSM is incapable of exploiting smaller values of p.

As before, let n be a composite natural number having no *small* prime divisors and let p be the smallest prime divisor of n. For denoting subexponential expressions in $\ln p$, we use the symbol $L_p[c] := L(p, 1/2, c)$, whereas the unsubscripted symbol $L[c]$ stands for $L(n, 1/2, c)$. We work with random elliptic curves

$$E : Y^2 = X^3 + aX + b,\ a, b \in \mathbb{Z}$$

and consider the group $G := E(\mathbb{Z}_p)$ of rational points on E modulo p. However, since p is not known a priori, we intend to work modulo n. The canonical surjection $\mathbb{Z}_n \to \mathbb{Z}_p$ allows us to identify the $\mathbb{Z}_n$-rational points on E as points on E over $\mathbb{Z}_p$. We now define a bound $M = L_p[1/\sqrt{2}]$ and let $B = \{q_1, \ldots, q_t\}$ be all the primes

[2] very large-scale integration

smaller than or equal to M, so that by the prime number theorem (Theorem 2.20) $\#B \approx M/\ln M = L_p[1/\sqrt{2}]$. Of course, p is not known in advance, so that M and B are also not known. We will discuss about the choice of M and B later. For the time being, let us assume that we know some approximate value of p, so that M and B can be fixed, at least approximately, at the beginning of the algorithm.

By Hasse's theorem (Theorem 2.48, p106), the cardinality $\nu := \#E(\mathbb{Z}_p)$ satisfies $|p + 1 - \nu| \leqslant 2\sqrt{p}$, that is, $\nu = \mathrm{O}(p)$. If we make the heuristic assumption that ν is a random integer on the order $\mathrm{O}(p)$, then Corollary 4.1 tells us that ν is B-smooth with probability $L_p[-1/\sqrt{2}]$. This assumption is certainly not rigorous, but accepting it gives us a way to analyse the running time of the algorithm.

If $L_p[1/\sqrt{2}]$ random curves are tried, then we expect to find one B-smooth value of ν. In this case, a non-trivial factor of n can be computed with high probability as follows. Define $e_i := \lfloor \ln n / \ln q_i \rfloor$ for $i = 1, \ldots, t$, and $m := \prod_{i=1}^{t} q_i^{e_i}$, where t is the number of primes in B. If ν is B-smooth, then $\nu | m$ and, therefore, for any point $P \in E(\mathbb{Z}_p)$ we have $mP = \mathcal{O}$. Computation of mP involves computation of many sums $P_1 + P_2$ of points $P_1 := (h_1, k_1)$ and $P_2 := (h_2, k_2)$. At some point of time, we would certainly compute $P_1 + P_2 = \mathcal{O}$, that is, $P_1 = -P_2$, that is, $h_1 \equiv h_2 \pmod{p}$ and $k_1 \equiv -k_2 \pmod{p}$. Since p was unknown, we worked modulo n, that is, the values of h_1, h_2, k_1 and k_2 are known modulo n. Let $d := \gcd(h_1 - h_2, n)$. Then $p|d$ and if $d \neq n$ (the case $d = n$ has a very small probability!), we have the non-trivial factor d of n. The computation of the coordinates of $P_1 + P_2$ (assuming $P_1 \neq P_2$) demands computing the inverse of $h_1 - h_2$ modulo n (Section 2.11.2). However, if $d = \gcd(h_1 - h_2, n) \neq 1$, then this inverse does not exist and so the computation of $P_1 + P_2$ fails, and we have a non-trivial factor of n. If ν is B-smooth, then the computation of mP is bound to fail. The basic steps of the ECM are then as shown in Algorithm 4.3.

Algorithm 4.3 **Elliptic curve method (ECM)**

Input: A composite integer $n \in \mathbb{N}$ (with no small prime factors).
Output: A non-trivial divisor d of n.
Steps:

```
while (1) {
  Select a random curve E : Y^2 = X^3 + aX + b modulo n.
  Choose a point P ≠ O in E(Z_n).
  Try to compute mP.   /* where m is as defined in the text */
  if (the computation of mP fails) {
    /* We have found a divisor d > 1 of n */
    if (d ≠ n) { Return d. }
  }
}
```

Before we derive the running time of the ECM, some comments are in order. A random curve E is chosen by selecting random integers a and b modulo n. It turns out that taking a as single-precision integers and $b = 1$ works quite well in practice.

Indeed one can keep on trying the values $a = 0, 1, 2, \ldots$ successively. Note that the curve E is an elliptic curve, that is, non-singular, if and only if $\delta := \gcd(n, 4a^3 + 27b^2) = 1$. However, having $\delta > 1$ is an extremely rare occurrence and one might skip the computation of δ before starting the trial with a curve. The choice $b = 1$ is attractive, because in that case we may take the point $P = (0, 1)$. In Section 3.6, we have described a strategy to find a random point on an elliptic curve over a field K. This is based on the assumption that computing square roots in K is easy. The same method can be applied to curves over $\mathbb{Z}_n$, but n being composite, it is difficult to compute square roots modulo n. So taking b to be 1 (or the square of a known integer) is indeed a pragmatic decision. After all, we do not need P to be a random point on E.

Recall that we have taken $m = \prod_{i=1}^{t} q_i^{e_i}$, where $e_i = \lfloor \ln n / \ln q_i \rfloor$. If instead we take $e_i := \lfloor \ln M / \ln q_i \rfloor$ (where M is the bound mentioned earlier), the computation of mP per trial reduces much, whereas the probability of a successful trial (that is, a failure of computing mP) does not decrease much. The integer m can be quite large. One, however, need not compute m explicitly, but proceed as follows: first take $Q_0 := P$ and subsequently for each $i = 1, \ldots, t$ compute $Q_i := q_i^{e_i} Q_{i-1}$. One finally gets $mP = Q_t$.

Now comes the analysis of the running time of the ECM. We have fixed the parameter M to be $L_p[1/\sqrt{2}]$, so that B contains $L_p[1/\sqrt{2}]$ small primes. The most expensive part of a trial with a random elliptic curve is the (attempted) computation of the point mP. This involves $\mathrm{O}(\ln m) = \mathrm{O}(L_p[1/\sqrt{2}] \ln n) = L_p[1/\sqrt{2}]$ additions of points. Since an expected number of $L_p[1/\sqrt{2}]$ elliptic curves needs to be tried for finding a non-trivial factor of n, the algorithm performs an expected number of $L_p[1/\sqrt{2}]L_p[1/\sqrt{2}] = L_p[\sqrt{2}]$ additions of points on curves modulo n. Since each such addition can be done in polynomial time, the announced running time follows.

Note that $L_p[\sqrt{2}]$ is the optimal running time of the ECM and can be shown to be achieved by taking $M = L_p[1/\sqrt{2}]$. But, in practice, p is not known a priori. Various *ad hoc* ways may be adopted to get around with this difficulty. One possibility is to use the worst-case bound $p \leqslant \sqrt{n}$. For example, for factoring integers of the form $n = pq$, where p and q are primes of roughly the same size, this is a good approximation for p. Another strategy is to start with a small value of M and increase M gradually with the number of trials performed. For larger values of M, the probability of a successful trial increases implying that less number of elliptic curves needs to be tried, whereas the time per iteration (that is, for the computation of mP) increases. In other words, the total running time of the ECM is apparently not very sensitive to the choice of M.

A *second stage* can be used for each elliptic curve in order to increase the probability of a trial being successful. A strategy very similar to the second stage of the $p - 1$ method can be employed. The reader is urged to fill out the details. Employing the second stage leads to reasonable speed-up in practice, though it does not affect the asymptotic running time.

The ECM can be effectively parallelized, since different processors can carry out the trials, that is, computations of mP (together with the second stage) with different sets of (random) elliptic curves.

** 4.3.4 The Number Field Sieve Method

The *number field sieve method* (*NFSM*) is till date the most successful of all integer factoring algorithms. Under certain heuristic assumptions it achieves a running time of the form $L(n, 1/3, c)$, which is better than the $L(n, 1/2, c')$ algorithms described so far. The NFSM was first designed for integers of a special form. This variant of the NFSM is called the *special NFS method* (*SNFSM*) and was later modified to the *general NFS method* (*GNFSM*) that can handle arbitrary integers. The running time of the SNFSM has $c = (32/9)^{1/3} \approx 1.526$, whereas that for the GNFSM has $c = (64/9)^{1/3} \approx 1.923$. For the sake of simplicity, we describe only the SNFSM in this book (see Cohen [56] and Lenstra and Lenstra [165] for further details).

We choose an integer $m \in \mathbb{N}$ and a polynomial $f(X) \in \mathbb{Z}[X]$ such that $f(m) \equiv 0 \pmod{n}$. We assume that f is irreducible in $\mathbb{Z}[X]$; otherwise a non-trivial factor of f yields a non-trivial factor of n. Consider the number field $K := \mathbb{Q}[X]/\langle f(X)\rangle$. Let $d := \deg f$ be the degree of the number field K. We use the complex embedding $K \cong \mathbb{Q}(\alpha)$ for some root $\alpha \in \mathbb{C}$ of f. The special NFS method makes certain simplifying assumptions:

(1) f is monic, so that $\alpha \in \mathfrak{O}_K$.
(2) $\mathfrak{O}_K = \mathbb{Z}[\alpha]$ is monogenic.
(3) $\mathfrak{O}_K$ is a PID.

Consider the ring homomorphism

$$\Phi : \mathfrak{O}_K = \mathbb{Z}[\alpha] \to \mathbb{Z}_n, \quad \alpha \mapsto m \pmod{n}.$$

This is well-defined, since $f(m) \equiv 0 \pmod{n}$. We choose *small* coprime (rational) integers a, b and note that $\Phi(a+b\alpha) = a+bm \pmod{n}$. Let $B \in \mathbb{N}$ be a predetermined *smoothness bound*. Assume that for a given pair (a, b), both $a + bm$ and $a + b\alpha$ are B-smooth. For the rational integer $a + bm$, this means

$$a + bm = \prod_{p \in \mathcal{P}} p^{v_p},$$

$\mathcal{P}$ being the set of all rational primes $\leqslant B$. On the other hand, smoothness of the algebraic integer $a + b\alpha$ means that the principal ideal $\langle a + b\alpha\rangle = (a + b\alpha)\mathfrak{O}_K$ is a product of prime ideals of prime norms $\leqslant B$; that is, we have a factorization

$$\langle a + b\alpha\rangle = \prod_{\mathfrak{q} \in \mathcal{Q}} \mathfrak{q}^{w_\mathfrak{q}},$$

where $\mathcal{Q}$ is the set of all prime ideals of $\mathfrak{O}_K$ of prime norms $\leqslant B$. By assumption, each $\mathfrak{q} \in \mathcal{Q}$ is a principal ideal. Let $\mathcal{G}$ denote a set of generators, one for each ideal in $\mathcal{Q}$. Further let $\mathcal{U}$ denote a set of generators of the multiplicative group of units of $\mathfrak{O}_K$. The smoothness of $a + b\alpha$ can, therefore, be rephrased as

$$a + b\alpha = \prod_{u \in \mathcal{U}} u^{r_u} \prod_{g \in \mathcal{G}} g^{w_g}. \tag{4.4}$$

Applying Φ then yields

$$\prod_{u \in \mathcal{U}} \Phi(u)^{r_u} \prod_{g \in \mathcal{G}} \Phi(g)^{w_g} \equiv \prod_{p \in \mathcal{P}} p^{v_p} \pmod{n}.$$

This is a *relation* for the SNFSM. After $\#\mathcal{U} + \#\mathcal{G} + \#\mathcal{P}$ relations are available, Gaussian elimination modulo 2 (as in the case of the QSM) is expected to give us a congruence of the form

$$x^2 \equiv y^2 \pmod{n},$$

and $\gcd(x - y, n)$ is possibly a non-trivial factor of n. This is the basic strategy of the SNFSM. We clarify some details now.

Selecting the polynomial $f(X)$

There is no clearly specified way to select the polynomial f for defining the number field $K = \mathbb{Q}[X]/\langle f \rangle$. We require f to have *small* coefficients. Typically, m is much smaller than n and one writes the expansion of n in base m as $n = b_t m^t + b_{t-1} m^{t-1} + \cdots + b_1 m + b_0$ with $0 \leqslant b_i < m$. Taking $f(X) = b_t X^t + b_{t-1} X^{t-1} + \cdots + b_1 X + b_0$ is often suggested.

For integers n of certain special forms, we have *natural* choices for f. The seminal paper on the NFSM by Lenstra et al. [167] assumes that $n = r^e - s$ for a small integer $r \in \mathbb{N}$ and a non-zero integer s with small absolute value. In this case, one first chooses a small extension degree $d \in \mathbb{N}$ and sets $m := r^{\lceil e/d \rceil}$ and $f(X) := X^d - s r^{\lceil e/d \rceil d - e}$. Typically, $d = 5$ works quite well in practice. Lenstra et al. report the implementation of the SNFSM for factoring $n = 3^{239} - 1$. The parameters chosen are $d = 5$, $m = 3^{48}$ and $f(X) = X^5 - 3$. In this case, $\mathfrak{O}_K = \mathbb{Z}[3^{1/5}]$ is monogenic and a PID.

Construction of $\mathcal{Q}$

Take a small rational prime $p \in \mathcal{P}$. From Section 2.13, it follows that if $\bar{f}(X) = \bar{f}_1(X)^{e_1} \cdots \bar{f}_r(X)^{e_r}$ is the factorization of the canonical image of $f(X)$ modulo p, then $\mathfrak{q}_i := \langle p, f_i(\alpha) \rangle \subseteq \mathfrak{O}_K$, $i = 1, \ldots, r$, are all the primes lying over p. We have also seen that $\mathrm{N}(\mathfrak{q}_i) = p^{d_i}$, $d_i := \deg \bar{f}_i$, is prime if and only if $d_i = 1$, that is, $\bar{f}_i(X) = X - c_p$ for some $c_p \in \mathbb{F}_p$. Thus, each root of $\bar{f}(X)$ in $\mathbb{F}_p$ corresponds to a prime ideal of $\mathfrak{O}_K$ of prime norm p.

To sum up, a prime ideal in $\mathfrak{O}_K$ of prime norm is specified by a pair (p, c_p) of values (in $\mathbb{Z} \times \mathbb{F}_p$). We denote this ideal by $\mathfrak{q}_{p,c_p}$. All ideals in $\mathcal{Q}$ can be precomputed by finding the roots of the defining polynomial $f(X)$ modulo the small primes $p \leqslant B$. One can use the root-finding algorithms of Exercise 3.29.

Construction of $\mathcal{G}$

Constructing a set $\mathcal{G}$ of generators of ideals in $\mathcal{Q}$ is a costly operation. We have just seen that each prime ideal $\mathfrak{q}$ of $\mathcal{Q}$ corresponds to a pair (p, c_p) and is a principal ideal by assumption. A generator $g_{p,c_p} = g_{\mathfrak{q}}$ of such an ideal $\mathfrak{q}$ is an element of the form $g_{p,c_p} = \sum_{i=0}^{d-1} g_i \alpha^i$, $g_i \in \mathbb{Z}$, with $\mathrm{N}(g_{p,c_p}) = \pm p$ and $\sum_{i=0}^{d-1} g_i c_p^i \equiv 0 \pmod{p}$. Algorithm 4.4 (quoted from Lenstra et al. [167]) computes the generators g_{p,c_p} for all relevant pairs (p, c_p). The first for loop exhaustively searches over all *small* polynomials $h(\alpha)$ in order to locate for each (p, c_p) an element $\bar{g}_{p,c_p}$ of norm kp with $|k|$ as small as possible. If the smallest k (stored in a_{p,c_p}) is ± 1, $\bar{g}_{p,c_p}$ is already a generator g_{p,c_p} of $\mathfrak{q}_{p,c_p}$, else some additional adjustments need to be performed.

Algorithm 4.4 **Construction of generators of ideals for the SNFSM**

Choose two suitable positive constants a_B and C_B (depending on B and K).
Initialize an array $a_{p,c_p} := a_B$ indexed by the relevant pairs (p, c_p).
for each $h = \sum_{i=0}^{d-1} h_i \alpha^i$, $h_i \in \mathbb{Z}$, with $\sum_{i=0}^{d-1} h_i^2 |\alpha|^{2i} \leqslant C_B$, $\mathrm{N}(h) = kp$, $p \leqslant B$,
  $k \in \mathbb{Z} \setminus \{0\}$, $|k| < \min(p, a_B)$ {
  Find c_p such that $\sum_{i=0}^{d-1} h_i c_p^i \equiv 0 \pmod{p}$. /* Root finding */
  if $(|k| < |a_{p,c_p}|)$ {
    /* Store the least k and the corresponding h found so far */
    $a_{p,c_p} := k$, $\bar{g}_{p,c_p} := h$.
  }
}
for each relevant pair (p, c_p) {
  if $(a_{p,c_p} = \pm 1)$ { $g_{p,c_p} := \bar{g}_{p,c_p}$. } /* The more frequent case */
  else {
    Locate a $\gamma \in \mathfrak{O}_K$ with $\mathrm{N}(g) = a_{p,c_p}$.
    $g_{p,c_p} := \bar{g}_{p,c_p}/\gamma$.
  }
}

Construction of $\mathcal{U}$

Let K have the signature (r_1, r_2). Write $\rho = r_1 + r_2 - 1$. By Dirichlet's unit theorem, the group $\mathfrak{U}_K$ of units of $\mathfrak{O}_K$ is generated by an appropriate root u_0 of unity and ρ multiplicatively independent[3] elements $u_1, \ldots, u_\rho$ of infinite order. Each unit u of $\mathfrak{O}_K$ has norm $\mathrm{N}(u) = \pm 1$. Thus, one may keep on generating elements $\sum_{i=0}^{d-1} h_i \alpha^i$, h_i small integers, of norm ± 1, until ρ independent elements are found. Many elements of $\mathcal{U}$ are available as a by-product during the construction of $\mathcal{G}$, which involves the computation of norms of many elements in $\mathfrak{O}_K$. For a more general exposition on this topic, see Algorithm 6.5.9 of Cohen [56].

Computing the factorization of $a + b\alpha$

In order to compute the factorization of Equation (4.4), we first factor the integer $\mathrm{N}(a + b\alpha) = b^d f(-a/b)$. If $\langle a + b\alpha \rangle = \mathfrak{q}_1^{w_1} \cdots \mathfrak{q}_r^{w_r}$ is the prime factorization of $\langle a + b\alpha \rangle$ with pairwise distinct prime ideals $\mathfrak{q}_1, \ldots, \mathfrak{q}_r$ of $\mathfrak{O}_K$, by the multiplicative property of norms we obtain $\mathrm{N}(a + b\alpha) = \mathrm{N}(\mathfrak{q}_1)^{w_1} \cdots \mathrm{N}(\mathfrak{q}_r)^{w_r}$.

Now, let $p \leqslant B$ be a small prime. If $p \nmid \mathrm{N}(a + b\alpha)$, it is clear that no prime ideal of $\mathfrak{O}_K$ of norm p (or a power of p) appears in the factorization of $\langle a + b\alpha \rangle$. On the other hand, if $p | \mathrm{N}(a + b\alpha)$, then $p | \mathrm{N}(\mathfrak{q}_i)$ for some $i \in \{1, \ldots, r\}$. The assumption $\mathfrak{O}_K = \mathbb{Z}[\alpha]$ implies that the inertial degree of $\mathfrak{q}_i$ is 1: that is, $\mathrm{N}(\mathfrak{q}_i) = p$, that is, $\mathfrak{q}_i \in \mathcal{Q}$, that is, there is a c_p with $f(c_p) \equiv 0 \pmod{p}$ such that the prime ideal $\mathfrak{q}_i$ corresponds to

[3] The elements $u_1, \ldots, u_\rho$ in a (multiplicatively written) group are called (*multiplicatively*) *independent* if $u_1^{n_1} \cdots u_\rho^{n_\rho}$, $n_i \in \mathbb{Z}$, is the group identity only for $n_1 = \cdots = n_\rho = 0$.

the pair (p, c_p). In this case, we have $a \equiv -c_p b \pmod{p}$. Assume that another prime ideal $\mathfrak{q}_j$ of norm p appears in the prime factorization of $\langle a + b\alpha \rangle$. If $\mathfrak{q}_j$ corresponds to the pair p, c'_p, then $a \equiv -c'_p b \pmod{p}$. Since c_p and c'_p are distinct modulo p, it follows that $p | \gcd(a, b)$, a contradiction, since $\gcd(a, b) = 1$. Thus, a unique ideal $\mathfrak{q} = \mathfrak{q}_i$ of norm p appears in the factorization of $\langle a + b\alpha \rangle$. Moreover, the multiplicity of $\mathfrak{q}$ in the factorization of $\langle a + b\alpha \rangle$ is the same as the multiplicity $\mathrm{v}_p(\mathrm{N}(a + b\alpha))$.

Thus, one may attempt to factorize $\mathrm{N}(a + b\alpha)$ using trial divisions by primes $\leqslant B$. If the factorization is successful, that is, if $\mathrm{N}(a + b\alpha)$ is B-smooth, then for each prime divisor p of $\mathrm{N}(a + b\alpha)$ we find out the ideal $\mathfrak{q}$ and its multiplicity in the factorization of $\langle a + b\alpha \rangle$ as explained above. Since we know a generator of each $\mathfrak{q} \in \mathcal{Q}$, we eventually compute a factorization $a + b\alpha = u \prod_{g \in \mathcal{G}} g^{w_g}$, where u is a unit in $\mathfrak{O}_K$. What remains is to factor u as a product of elements of $\mathcal{U}$. We don't discuss this step here, but refer the reader to Lenstra et al. [167].

Sieving

In the QSM, we check the smoothness of a single integer $T(c)$ per trial, whereas for the NFS method we do so for two integers, namely, $a + bm$ and $\mathrm{N}(a + b\alpha)$. However, both these integers are much smaller than $T(c)$, and the probability that they are simultaneously smooth is larger than the probability that $T(c)$ alone is smooth. This accounts for the better asymptotic performance of the NFS method compared to the QSM.

One has to check the smoothness of $a + bm$ and $\mathrm{N}(a + b\alpha)$ for each coprime a, b in a predetermined interval. This check can be carried out efficiently using sieves. We have to use two sieves, one for filtering out the non-smooth $a + bm$ values and the other for filtering out the non-smooth $a + b\alpha$ values. We should have $\gcd(a, b) = 1$, but computing $\gcd(a, b)$ for all values of a and b is rather costly. We may instead use a third sieve to throw away the values of a for a given b for which $\gcd(a, b)$ is divisible by primes $\leqslant B$. This still leaves us with some pairs (a, b) for which $\gcd(a, b) > 1$. But this is not a serious problem, since such values are small in number and can be later discarded from the list of pairs (a, b) selected by the smoothness test.

We fix b and allow a to vary in the interval $-M \leqslant a \leqslant M$ for a predetermined bound M. We use an array $\mathfrak{A}$ indexed by a. Before the first sieve we initialize this array to $\mathfrak{A}_a := \ln |a + mb|$. We may set $\mathfrak{A}_a := +\infty$ for those values of a for which $\gcd(a, b)$ is known to be > 1 (where $+\infty$ stands for a suitably large positive value). For each small prime $p \leqslant B$ and small exponent $h \in \mathbb{N}$, we compute $a' := -mb \pmod{p^h}$ and subtract $\ln p$ from $\mathfrak{A}_a$ for each a, $-M \leqslant a \leqslant M$, with $a \equiv a' \pmod{p^h}$. Finally, for each value of a for which $\mathfrak{A}_a$ is not (close to) 0, that is, for which $a + mb$ is not B-smooth, we set $\mathfrak{A}_a := +\infty$. For the other values of a, we set $\mathfrak{A}_a := \ln |\mathrm{N}(a + b\alpha)|$. One may use incomplete sieving (with a liberal selection criterion) during the first sieve.

The second sieve proceeds as follows. We continue to work with the value of b fixed before the first sieve and with the array $\mathfrak{A}$ available from the first sieve. For each prime ideal $\mathfrak{q} = \mathfrak{q}_{p,c_p} \in \mathcal{Q}$, we compute $a'' := -bc_p \pmod{p}$ and subtract $\ln p$ from each location $\mathfrak{A}_a$ for which $a \equiv a'' \pmod{p}$. For those a for which $\mathfrak{A}_a \leqslant \xi \ln B$ for some real $\xi \geqslant 1$, say $\xi = 2$, we try to factorize $a + b\alpha$ over $\mathcal{G}$ and $\mathcal{U}$. If the attempt is successful, both $a + bm$ and $a + b\alpha$ are smooth. This second sieve is an incomplete one and, therefore, we must use a liberal selection criterion.

The running time of the SNFSM

For deriving the running time of the SNFSM, take $d \leqslant (3\ln n/(2\ln\ln n))^{1/3}$, $m = L(n, 2/3, (2/3)^{1/3})$, $B = L(n, 1/3, (2/3)^{2/3})$ and $M = L(n, 1/3, (2/3)^{2/3})$. From the prime number theorem and from the fact that d is small, it follows that both $\#\mathcal{P}$ and $\#\mathcal{G}$ have the same asymptotic bound as B. Also $\#\mathcal{U} \leqslant d$ meets this bound. We then have $L(n, 1/3, (2/3)^{2/3})$ unknown quantities on which we have to do Gaussian elimination.

The integers $a + mb$ have absolute values $\leqslant L(n, 2/3, (2/3)^{1/3})$. If the coefficients of f are small, then

$$|\,\mathrm{N}(a + b\alpha)| = |b^d f(-a/b)| \leqslant L(n, 1/3, d \cdot (2/3)^{2/3}) = L(n, 2/3, (2/3)^{1/3}).$$

Under the *heuristic assumption* that $a + mb$ and $\mathrm{N}(a + b\alpha)$ behave as random integers of magnitude $L(n, 2/3, (2/3)^{1/3})$, the probability that both these are B-smooth turns out to be $L(n, 1/3, -(2/3)^{2/3})$, and so trying $L(n, 1/3, 2(2/3)^{2/3})$ pairs (a, b) is expected to give us $L(n, 1/3, (2/3)^{2/3})$ relations. The entire sieving process takes time $L(n, 1/3, 2(2/3)^{2/3})$, whereas solving a sparse system in $L(n, 1/3, (2/3)^{2/3})$ unknowns can be done essentially in the same time. Thus the running time of the SNFSM is $L(n, 1/3, 2(2/3)^{2/3}) = L(n, 1/3, (32/9)^{1/3})$.

EXERCISE SET 4.3

4.6 For $m \in \mathbb{N}$, define the *harmonic numbers* $H_m := \frac{1}{1} + \frac{1}{2} + \cdots + \frac{1}{m} \in \mathbb{Q}$. Show that for each $m \in \mathbb{N}$ we have $\ln(m + 1) \leqslant H_m \leqslant 1 + \ln m$. **[H]** Deduce that the sequence H_m, $m \in \mathbb{N}$, is not convergent. (Note, however, that the sequence $H_m - \ln m$, $m \in \mathbb{N}$, converges to the constant $\gamma = 0.57721566\ldots$ known as the *Euler constant*. It is not known whether γ is rational or not.)

4.7 Let k, c, c', α be positive constants with $\alpha < 1$. Prove the following assertions.

(a) $L(n, \alpha, c) = n^{[(c+\mathrm{o}(1))\left(\frac{\ln\ln n}{\ln n}\right)^{1-\alpha}]}$.
(b) $L(n, \alpha, c)L(n, \alpha, c')$ is of the form $L(n, \alpha, c + c')$.
(c) $(\ln n)^k L(n, \alpha, c)$ is again of the form $L(n, \alpha, c)$.
(d) $L(n, \alpha, c)n^k$ is of the form $n^{k+\mathrm{o}(1)}$.

4.8 Let us assume that an adversary C has computing power to carry out 10^{12} floating point operations (flops) per second. Let A be an algorithm that computes a certain function $P(n)$ using $T(n)$ flops for an input $n \in \mathbb{N}$. We say that it is *infeasible* for C to compute $P(n)$ using algorithm A, if it takes $\geqslant 100$ years for the computation or, equivalently, if $T(n) \geqslant 3.1536 \times 10^{21}$. Find, for the following expressions of $T(n)$, the smallest values of n that make the computation of $P(n)$ by Algorithm A infeasible: $T(n) = (\ln n)^3$, $T(n) = (\ln n)^{10}$, $T(n) = n$, $T(n) = \sqrt{n}$, $T(n) = n^{1/4}$, $T(n) = L[2]$, $T(n) = L[1]$, $T(n) = L[0.5]$, $T(n) = L(n, 1/3, 2)$ and $T(n) = L(n, 1/3, 1)$. (Neglect the $\mathrm{o}(1)$ terms in the definitions of $L(\,)$ and $L[\,]$.)

4.9 Let $n \in \mathbb{N}$ be an odd integer and let r be the total number of distinct (odd) prime divisors of n. Show that for each integer a the congruence $x^2 \equiv a^2 \pmod{n}$ has $\leqslant 2^r$

solutions for x modulo n. If $\gcd(a, n) = 1$, show that this congruence has exactly 2^r solutions. **[H]**

4.10 Show that the problems IFP and SQRTP are probabilistic polynomial-time equivalent. **[H]**

4.11 In this exercise, we use the notations introduced in connection with the Quadratic Sieve method for factoring integers (Section 4.3.2). We assume that $M \ll H$, since $H = \mathrm{O}(\sqrt{n})$, whereas $M = L[1]$.
(a) Show that $J \leqslant 2H - 1$.
(b) Prove that the average of the integers $|T(c)|$, $-M \leqslant c \leqslant M$, is $MH\left(\frac{2M+2}{2M+1}\right) \approx MH$ and that the maximum of the same integers is $|T(M)| = J + 2MH + M^2 \approx J + 2MH$.
(c) Prove that the average and the maximum of the integers $|T(c)|$, $0 \leqslant c \leqslant 2M$, are respectively $J + 2MH + M(4M + 1)/3 \approx J + 2MH$ and $|T(2M)| = J + 4MH + 4M^2 \approx J + 4MH$.
(d) Conclude that it is better to choose the sieving interval as $-M \leqslant c \leqslant M$ instead of as $0 \leqslant c \leqslant 2M$.

4.12 Reyneri's cubic sieve method (CSM) Suppose that we want to factor an odd integer n. Suppose also that we *know* a triple (x, y, z) of integers satisfying $x^3 \equiv y^2 z \pmod{n}$ with $x^3 \neq y^2 z$ (as integers). We assume further that $|x|, |y|, |z|$ are all $\mathrm{O}(p^\xi)$ for some ξ, $1/3 < \xi < 1/2$.
(a) Show that for integers a, b, c with $a + b + c = 0$ one has

$$(x + ay)(x + by)(x + cy) \equiv y^2 T(a, b, c) \pmod{n},$$

where $T(a, b, c) := z + (ab + ac + bc)x + (abc)y = -b(b + c)(x + cy) + (z - c^2 x)$. If $x, y, z = \mathrm{O}(p^\xi)$, then $T(a, b, c)$ is $\mathrm{O}(p^\xi)$ for small values of a, b, c.
(b) Let $\alpha := \sqrt{\xi/2}$. Choose a factor base comprising all primes $q_1, \ldots, q_t$ with $t = L[\alpha]$ together with the integers $x + ay$, $-M \leqslant a \leqslant M$, $M = L[\alpha]$. The size of the factor base is then $L[\alpha]$.
If $T(a, b, c)$ with $-M \leqslant a, b, c \leqslant M$ and $a + b + c = 0$ is q_t-smooth, we get a *relation* for the CSM. Show that trying out the $L[2\alpha]$ pairs (a, b, c) gives us a set of linear congruences of the desired size under the heuristic assumption that the $T(a, b, c)$ values behave as random integers on the order of p^ξ.
(c) Propose a strategy how these linear congruences can be combined (by Gaussian elimination) to get a quadratic congruence of the form $u^2 \equiv v^2 \pmod{n}$.
(d) Design a sieve for checking the smoothness of the expressions $T(a, b, c)$. **[H]**
(e) Show that the running time of the CSM is $L[2\alpha] = L[\sqrt{2\xi}]$. Since $\xi < 1/2$, the CSM is more efficient than the QSM. For $\xi \approx 1/3$, the running time is $L[\sqrt{2/3}] \approx L[0.816]$.
(**Remark:** It is not known how we can efficiently obtain a solution of $x^3 \equiv y^2 z \pmod{n}$ with $x^3 \neq y^2 z$ and $|x|, |y|, |z| = \mathrm{O}(p^\xi)$, ξ being as small as possible. For some particular values of n, say, for n of the form $x^3 - z$ with small z, a solution is naturally available.)

4.13 Sieve of Eratosthenes Two hundred years before Christ, Eratosthenes proposed a sieve (Algorithm 4.5) for computing all primes between 1 and a positive integer n. Prove the correctness of this algorithm and compute its running time. **[H]**

Algorithm 4.5 **The sieve of Eratosthenes**

Initialize to zero an array A indexed $2, \ldots, n$.
for $k = 2, \ldots, \lfloor \sqrt{n} \rfloor$ {
  if ($A_k = 0$) { for $l = 2, \ldots, \lfloor n/k \rfloor$ { $A_{lk} := 1$. } }
}
for $k = 2, \ldots, n$ { if ($A_k = 0$) { Print "k is a prime". } }

4.14 This exercise proposes an adaptation of the sieve of Eratosthenes for computing a random prime of a given bit length l. In Section 3.4.2, we have described an algorithm for this computation, that generates random (odd) integers of bit length l and checks the primality of each such integer, until a (probable) prime is found. An alternative strategy is to generate a random l-bit odd integer n and check the integers $n, n+2, n+4, \ldots$ for primality.

(a) Use sieving to design an algorithm that generalizes this second strategy in the sense that it checks for primality only those integers $n + r$, $r = 0, 1, 2, \ldots, M$, n a random l-bit integer, which are not divisible by the first t primes. In practice, the values $100 \leqslant t \leqslant 10{,}000$ and $M = 10l$ work quite well. For cryptographic sizes, sieving typically speeds up the generation of naive primes 10 to 100 times.

(b) Generalize the sieve of Part (a) for the computation of safe and strong primes.

4.4 The Finite Field Discrete Logarithm Problem

The discrete logarithm problem (DLP) has attracted somewhat less attention of the research community than the IFP has done. Nonetheless, many algorithms exist to solve the DLP, most of which are direct adaptations of algorithms for solving the IFP. We start with the older algorithms collectively known as the *square root methods*, since the worst-case running time of each of these is $\tilde{O}(\sqrt{q})$ for the field $\mathbb{F}_q$. The new family of algorithms based on the *index calculus method* provides subexponential solutions to the DLP and is described next. For the sake of simplicity, we assume in this section that we want to compute the discrete logarithm $\mathrm{ind}_g\, a$ of $a \in \mathbb{F}_q^*$ with respect to a *primitive element* g of $\mathbb{F}_q^*$. We concentrate only on the fields $\mathbb{F}_p$, p odd prime, and $\mathbb{F}_{2^n}$, $n \in \mathbb{N}$, since non-prime fields of odd characteristics are only rarely used in cryptography.

4.4.1 Square Root Methods

Square root methods are applicable to any finite (cyclic) group. To avoid repetitions we provide here a generic description. That is, we assume that G is a multiplicatively written group of order n and $g \in G$. The identity of G is denoted as 1. It is not

necessary to assume that G is cyclic or that g is a generator of G. However, these assumptions are expected to make the descriptions of the algorithms somewhat easier and hence we will stick to them. The necessary modifications for non-cyclic groups G or non-primitive elements g are rather easy and the reader is requested to fill out the details. We assume that each element of G can be represented by $\mathrm{O}(\lg n)$ bits (so that the input size is taken to be $\lg n$) and that multiplications, exponentiations and inverses in G can be computed in time polynomially bounded by this input size.

Shanks' baby-step–giant-step method

Let us assume that the elements of G can be (totally) ordered in such a way that comparing two elements of G with respect to this order can be done in polynomial time of the input size. For example, a natural order on $G = \mathbb{F}_p^* = \{1, \ldots, p-1\}$ is the relation $\leqslant$ on $\mathbb{Z}$. Note that k elements of G can be sorted (under the above order) using $\mathrm{O}(k \log k)$ comparisons.

Let $m := \lceil \sqrt{n} \rceil$. Then $d := \operatorname{ind}_g a$ is uniquely determined by two (nonnegative) integers $d_0, d_1 < m$ such that $d = d_0 + d_1 m$ (the base m representation of d). In Shanks' *baby-step–giant-step* (*BSGS*) method, we compute d_0 and d_1 as follows. To start with we compute a list of pairs (d_0, g^{d_0}) for $d_0 = 0, 1, \ldots, m-1$ and store these pairs in a table sorted with respect to the second coordinate (the *baby steps*). Now, for each $d_1 = 0, 1, \ldots, m-1$, we compute g^{-md_1} (the *giant steps*) and search if ag^{-md_1} is the second coordinate of a pair (d_0, g^{d_0}) of some entry in the table mentioned above. If so, we have found the desired d_0 and d_1, otherwise we try with the next value of d_1. An optimized implementation of this strategy is given as Algorithm 4.6.

The computation of all the elements of T and sorting T can be done in time $\tilde{\mathrm{O}}(m)$. If we use a binary search algorithm (Exercise 4.15), then the search for h in T can be performed using $\mathrm{O}(\lg m)$ comparisons in G. Therefore, the giant steps also take a total running time of $\tilde{\mathrm{O}}(m)$. Since $m = \lceil \sqrt{n} \rceil$, the BSGS method runs in time $\tilde{\mathrm{O}}(\sqrt{n})$. The memory requirement of the BSGS (that is, of the table T) is $\tilde{\mathrm{O}}(\sqrt{n})$. Thus this method becomes impractical, even when n contains as few as 30 decimal digits.

Pollard's rho method

Pollard's rho method for solving the DLP is similar in idea to the method with the same name for solving the IFP. Let $f : \mathbb{Z}_n^2 \to \mathbb{Z}_n^2$ be a random map and let us generate a sequence of tuples $(r_i, s_i) \in \mathbb{Z}_n^2$, $i \in \mathbb{N}$, starting with a random (r_1, s_1) and subsequently computing $(r_{i+1}, s_{i+1}) = f(r_i, s_i)$ for each $i = 1, 2, \ldots$. The elements $b_i := a^{r_i} g^{s_i} \in G$ for $i = 1, 2, \ldots$ can then be thought of as randomly chosen ones from G. By the birthday paradox (Exercise 2.172), we expect to get a match $b_i = b_j$ for some $i \neq j$, after $\mathrm{O}(\sqrt{n})$ of the elements $b_1, b_2, \ldots$ are generated. But then we have $a^{r_i - r_j} = g^{s_j - s_i}$, that is, $\operatorname{ind}_g a \equiv (r_i - r_j)^{-1}(s_j - s_i) \pmod{n}$, provided that the inverse exists, that is, $\gcd(r_i - r_j, n) = 1$. The expected running time of this algorithm is $\tilde{\mathrm{O}}(\sqrt{n})$, the same as that of the BSGS method, but the storage requirement drops to only $\mathrm{O}(1)$ elements of G.

Algorithm 4.6 **Shanks' baby-step–giant-step method**

Input: G, g and a as described above.
Output: $d = \operatorname{ind}_g a$.
Steps:
$n := \operatorname{ord}(G)$, $m := \lceil \sqrt{n} \rceil$.

/* Baby steps */
Initialize T to an empty table.
Insert the pairs $(0, 1)$ and $(1, g)$ in T.
$h := g$.
for $d_0 = 2, \ldots, m-1$ {
  $h := hg$.
  Insert (d_0, h) in T.
}
sort T with respect to the second coordinate.

/* Giant steps */
$h := a$. $l := (g^{-1})^m$.
for $d_1 = 0, \ldots, m-1$ {
  if T contains an entry (d_0, h) { Return $d := d_0 + d_1 m$. }
  $h := hl$.
}

The Pohlig–Hellman method

The Pohlig–Hellman (PH) method assumes that the prime factorization of $n = \operatorname{ord} G = p_1^{\alpha_1} \cdots p_r^{\alpha_r}$ is known. Since $d := \operatorname{ind}_g a$ is unique modulo n, we can easily compute d using the CRT from a knowledge of d modulo $p_j^{\alpha_j}$, $j = 1, \ldots, r$. So assume that p is a prime dividing n and that $\alpha := \mathrm{v}_p(n) \in \mathbb{N}$. Let $d_0 + d_1 p + \cdots + d_{\alpha-1} p^{\alpha-1}$, $d_i \in \{0, 1, \ldots, p-1\}$, be the p-ary representation of d modulo p^α. The p-ary digits $d_0, d_1, \ldots, d_{\alpha-1}$ can be successively computed as follows.

Let H be the subgroup of G generated by $h := g^{n/p}$. We have $\operatorname{ord} H = p$ (Exercise 2.44). For the computation of d_i, $i \in \{0, 1, \ldots, \alpha-1\}$, from the knowledge of $d_0, \ldots, d_{i-1}$, consider the element

$$\begin{aligned} b &:= \left[a \left(g^{d_0 + \cdots + d_{i-1} p^{i-1}} \right)^{-1} \right]^{n/p^{i+1}} \\ &= \left(g^{d - d_0 - \cdots - d_{i-1} p^{i-1}} \right)^{n/p^{i+1}} = \left(g^{n/p^{i+1}} \right)^{d - d_0 - \cdots - d_{i-1} p^{i-1}}. \end{aligned}$$

But $\operatorname{ord}(g^{n/p^{i+1}}) = p^{i+1}$, so that

$$b = \left(g^{(d \bmod p^{i+1}) - d_0 - \cdots - d_{i-1} p^{i-1}} \right)^{n/p^{i+1}} = \left(g^{d_i p^i} \right)^{n/p^{i+1}} = (g^{n/p})^{d_i} = h^{d_i}.$$

Thus, $b \in H$ and $d_i = \operatorname{ind}_h b$, that is, each d_i can be obtained by computing a discrete logarithm in the group H of order p (using the BSGS method or the rho method).

From the prime factorization of n, we see that the computations of d modulo $p_j^{\alpha_j}$ for all $j = 1, \ldots, r$ can be done in time $\tilde{\mathrm{O}}(\sum_{j=1}^{r} \alpha_j \sqrt{p_j}) = \tilde{\mathrm{O}}(\sqrt{q})$, q being the largest prime factor of n, since α_j and r are $\mathrm{O}(\log n)$. Combining the values of d modulo $p_j^{\alpha_j}$ by the CRT can be done in polynomial time (in $\log n$). In the worst case, $q = \mathrm{O}(n)$ and the PH method takes time $\tilde{\mathrm{O}}(\sqrt{n})$ which is fully exponential in the input size $\log n$. But if q (or, equivalently, all the prime divisors $p_1, \ldots, p_r$ of n) are *small*, then the PH method runs quite efficiently. In particular, if $q = \mathrm{O}((\log p)^c)$ for some (small) constant c, then the PH method computes discrete logarithms in G in *polynomial time*. This fact has an important bearing on the selection of a group G for cryptographic applications, namely, $n = \operatorname{ord} G$ is *required* to have a suitably large prime divisor, so that the PH method cannot compute discrete logarithms in G in feasible time.

4.4.2 The Index Calculus Method

The *index calculus method* (*ICM*) is not applicable to all (cyclic) groups. But whenever it applies, it usually leads to the fastest algorithms to solve the DLP. Several variants of the ICM are used for prime finite fields and also for finite fields of characteristic 2. On such a field $\mathbb{F}_q$ they achieve subexponential running times of the order of $L(q, 1/2, c) = L[c]$ or $L(q, 1/3, c)$ for positive constants c. We start with a generic description of the ICM. We assume that g is a primitive element of $\mathbb{F}_q^*$ and want to compute $d := \operatorname{ind}_g a$ for some $a \in \mathbb{F}_q^*$.

To start with we fix a *suitable* subset $B = \{b_1, \ldots, b_t\}$ of $\mathbb{F}_q^*$ of *small* cardinality, so that a *reasonably large* fraction of the elements of $\mathbb{F}_q^*$ can be expressed *easily* as products of elements of B. We call B a *factor base*. In the ICM, we search for *relations* of the form

$$g^{\alpha} a^{\beta} \prod_{i=1}^{t} b_i^{\gamma_i} = \prod_{i=1}^{t} b_i^{\delta_i} \quad (\text{in } \mathbb{F}_q^*), \tag{4.5}$$

for integers α, β, γ_i and δ_i. This gives us a linear congruence

$$\alpha + \beta \operatorname{ind}_g a + \sum_{i=1}^{t} \gamma_i \operatorname{ind}_g b_i \equiv \sum_{i=1}^{t} \delta_i \operatorname{ind}_g b_i \pmod{q-1}. \tag{4.6}$$

The ICM proceeds in two[4] stages. In the *first stage*, we compute $d_i := \operatorname{ind}_g b_i$ for each element b_i in the factor base B. For that, we collect Relation (4.5) with $\beta = 0$. When sufficiently many relations are available, the corresponding system of linear Congruences (4.6) is solved mod $q - 1$ for the unknowns d_i. In the *second stage*, a single relation with $\gcd(\beta, q-1) = 1$ is found. Substituting the values of d_i available from the first stage yields $\operatorname{ind}_g a$.

Note that as long as q (and g) are fixed, we don't have to carry out the first stage every time the discrete logarithm of an element of $\mathbb{F}_q^*$ is to be computed. If the values

[4] Some authors prefer to say that the number of stages in the ICM in actually three, because they decouple the congruence-solving phase from the first stage. This is indeed justified, since implementations by several researchers reveal that for large fields this linear algebra part often demands running time comparable to that needed by the relation collection part. Our philosophy is to call the entire *precomputation* work the *first stage*. Now, although it hardly matters, it is up to the reader which camp she wants to join.

d_i, $i = 1, \ldots, t$, are stored, then only the second stage needs to be carried out for computing the indices of any number of elements of $\mathbb{F}_q^*$. This is the reason why the first stage of the ICM is often called the *precomputation stage*.

In order to make the algorithm more concrete, we have to specify:

(1) how to choose a factor base B;
(2) how to find Relation (4.5);
(3) how to solve a linear system of congruences modulo $q-1$ (in particular, when the system is *sparse*).

In the rest of this section, we describe variants of the ICM based on their strategies for selecting the factor base and for collecting relations. We discuss the third issue in Section 4.7.

4.4.3 Algorithms for Prime Fields

Let $\mathbb{F}_p$ be a finite field of prime cardinality. For cryptographic applications, p should be quite large, say, of length around thousand bits or more, and so naturally p is odd. Elements of $\mathbb{F}_p$ are canonically represented as integers between (and including) 0 and $p-1$. The equality $x = y$ in $\mathbb{F}_p$ means equality of two integers in the range $0, \ldots, p-1$, whereas $x \equiv y \pmod{p}$ means that the two integers x and y may be different, but their equivalence classes in $\mathbb{Z}/\langle p \rangle = \mathbb{Z}_p = \mathbb{F}_p$ are the same.

The basic ICM

In the basic version of the ICM, we choose the factor base B to comprise the first t primes $q_1, \ldots, q_t$, where $t = L[\zeta]$. (The optimal value of ζ is determined below.) In the first stage, we choose random values of $\alpha \in \{1, \ldots, p-2\}$ and compute g^α. Any integer representing g^α can be considered, but we think of $g^\alpha \in \mathbb{F}_p$ as an integer in $\{1, \ldots, p-1\}$. We then try to factorize g^α using trial divisions by elements of the factor base B. If g^α is found to be B-smooth, then we get a desired relation for the first stage, namely,

$$g^\alpha = \prod_{i=1}^{t} q_i^{\delta_i} .$$

If g^α is not B-smooth, we try another random α and proceed as above. After sufficiently many relations are available, we solve the resulting system of linear congruences modulo $p-1$. This gives us $d_i := \operatorname{ind}_g q_i$ for $i = 1, \ldots, t$.

In the second stage, we again choose random integers α and try to factorize ag^α completely over B. Once the factorization gets successful, that is, we have $ag^\alpha = \prod_{i=1}^{t} q_i^{\epsilon_i}$, we compute $d = \operatorname{ind}_g a \equiv -\alpha + \sum_{i=1}^{t} \epsilon_i d_i \pmod{p-1}$.

In order to optimize the running time, we note that the relation collection phase of the first stage is usually the bottleneck of the algorithm. If ζ (or equivalently t) is chosen to be too small, then finding B-smooth integers would be very difficult. On the other hand, if ζ is too large, then we have to collect too many relations to have a solvable linear system of congruences. More explicitly, since the integers g^α can be

regarded as random integers of the order of p, the probability that g^α is B-smooth is $L[-\frac{1}{2\zeta}]$ (Corollary 4.1). Thus we expect to get each relation after $L[\frac{1}{2\zeta}]$ random values of α are tried. Since for each α we need to carry out $L[\zeta]$ divisions by elements of the factor base B (the exponentiation g^α can be done in polynomial time and hence can be neglected for this analysis), each relation can be found in expected time $L[\zeta + \frac{1}{2\zeta}]$. Now, in order to solve for d_i, $i = 1, \ldots, t$, we must have (slightly more than) $t = L[\zeta]$ relations. Thus, the relation collection phase takes a total time of $L[2\zeta + \frac{1}{2\zeta}]$. It can be easily checked that $2\zeta + \frac{1}{2\zeta}$ is minimized for $\zeta = 1/2$. This gives a running time of $L[2]$ for the relation collection phase.

Since each g^α is a positive integer less than p, it is evident that it can have at most $\mathrm{O}(\log p)$ prime divisors. In other words, the congruences collected are necessarily sparse. As we will see later, such a system can be solved in time $\tilde{\mathrm{O}}(t^2)$, that is, in time $L[1]$ for $\zeta = 1/2$.

In the second stage, it is sufficient to have a single relation to compute $d = \mathrm{ind}_g\, a$. As explained before, such a relation can be found in expected time $L[\zeta + \frac{1}{2\zeta}] = L[1.5]$. Thus the total running time of the basic ICM is $L[2]$.

The second stage of the basic ICM is much faster than the first stage. In fact, this is a typical phenomenon associated with most variants of the ICM. Speeding up the first stage is, therefore, our primary concern.

Each step in the search for relations consists of an exponentiation (g^α) modulo p followed by trial divisions by $q_1, \ldots, q_t$. Now, g^α may be non-smooth, but $g^\alpha + kp$ (integer sum) may be smooth for some $k \in \mathbb{Z}$. Once g^α is computed and found to be non-smooth, one can check for the smoothness of $g^\alpha + kp$ for $k = \pm 1, \pm 2, \ldots$, before another α is tried. Since these integers are available by addition (or subtraction) only (which is much faster than exponentiation), this strategy tends to speed up the relation collection phase. Moreover, information about the divisibility of $g^\alpha + kp$ by q_i can be obtained from that of $g^\alpha + (k-1)p$ by q_i. So using suitable tricks one might reduce the cost of trial divisions. Two such possibilities are explored in Exercise 4.18. Though these modifications lead to some speed-up in practice, they have the disadvantage that as $|k|$ increases, the size of $|g^\alpha + kp|$ also increases, so that the chance of getting smooth candidates reduces, and therefore using high values of k does not effectively help.

There are other heuristic modification schemes that help us gain some speed-up in practice. For example, the large prime variation as discussed in connection with the QSM applies equally well here. Another trick is to use the *early abort strategy*. A random B-smooth integer has higher probability of having many small prime factors rather than a few large prime factors. This observation can be incorporated in the smoothness tests as follows. Let us assume that we do trial divisions by the small primes in the order $q_1, q_2, \ldots, q_t$. After we do trial divisions of a candidate x by the first $t' < t$ primes (say, $t' \approx t/2$), we check how far we have been able to reduce x. If the reduction of x is already *substantial*, we continue with the trial divisions by the remaining primes $q_{t'+1}, \ldots, q_t$. In the other case, we abort the smoothness test for x and try another candidate. Obviously, this strategy prematurely rejects some smooth candidates (which are anyway rather small in number), but since most candidates are expected to be non-smooth, it saves a lot of trial divisions in the long run. Determination of t' and/or the quantization of "substantial" reduction actually depend on practical experiences. With suitable choices one may expect to get a speed-up of about 2. The drawback with the

early abort strategy is that it often does not go well with sieving. Sieving, whenever applicable, should be given higher preference.

To sum up, the basic ICM and all its modifications can be used for computing discrete logarithms only in small fields, say, of size $\leqslant 80$ bits. For bigger fields, we need newer ideas.

The linear sieve method

The *linear sieve method* (*LSM*) is a direct adaptation of the quadratic sieve method for factoring integers (Section 4.3.2). In the basic ICM just discussed, we try to find smooth integers from candidates that are on an average as large as $\mathrm{O}(p)$. The LSM, on the other hand, finds smooth ones from a pool of integers each of which is $\mathrm{O}(\sqrt{p})$. As a result, we expect to have a higher density of smooth integers among the candidates tested in the LSM than those in the basic method. Furthermore, the LSM employs sieving techniques instead of trial divisions. All these help LSM achieve a running time of $L[1]$, a definite improvement over the $L[2]$ performance of the basic method.

Let $H := \lfloor \sqrt{p} \rfloor + 1 = \lceil \sqrt{p} \rceil$ and $J := H^2 - p$. Then $J \leqslant 2\sqrt{p}$. Let's consider the congruence

$$(H + c_1)(H + c_2) \equiv J + (c_1 + c_2)H + c_1c_2 \pmod{p} \qquad \textbf{(4.7)}$$

For *small* integers c_1, c_2, the right side of the above congruence, henceforth denoted as

$$T(c_1, c_2) := J + (c_1 + c_2)H + c_1c_2 \,,$$

is of the order of $\sqrt{p}$. If the integer $T(c_1, c_2)$ is smooth with respect to the first t primes $q_1, q_2, \ldots, q_t$, that is, if we have a factorization like $J + (c_1 + c_2)H + c_1c_2 = \prod_{i=1}^{t} q_i^{\delta_i}$, then we have a relation

$$\operatorname{ind}_g(H + c_1) + \operatorname{ind}_g(H + c_2) = \sum_{i=1}^{t} \delta_i \operatorname{ind}_g q_i \,.$$

For the linear sieve method, the factor base comprises primes less than $L[1/2]$ (so that $t = L[1/2]$ by the prime number theorem) and integers $H + c$ for $-M \leqslant c \leqslant M$. The bound M on c is chosen to be of the order of $L[1/2]$. Each $T(c_1, c_2)$, being $\mathrm{O}(\sqrt{p})$ in absolute value, has a probability of $L[-1/2]$ for being q_t-smooth. Thus once we check the factorization of $T(c_1, c_2)$ for all (that is, for a total of $L[1]$) values of the pair (c_1, c_2) with $-M \leqslant c_1 \leqslant c_2 \leqslant M$, we expect to get $L[1/2]$ Relations (4.7) involving the unknown indices of the factor base elements. If we further assume that the primitive element g is a small prime which itself is in the factor base, then we get a *free* relation $\operatorname{ind}_g g = 1$. The resulting system is then solved to compute the discrete logarithms of elements in the factor base. This is the basic principle for the first stage of the LSM.

If we compute all $T(c_1, c_2)$ and use trial divisions by $q_1, \ldots, q_t$ to separate out the smooth ones, we achieve a running time of $L[1.5]$, as can be easily seen. Sieving is employed to reduce the running time to $L[1]$. First one fixes a $c_1 \in \{0, \pm 1, \ldots, \pm M\}$ and initializes to $\ln |T(c_1, c_2)|$ an array $\mathfrak{A}$ indexed by c_2 in the range $c_1 \leqslant c_2 \leqslant M$. One then computes for each prime power q^h, q being a small prime in the factor base and h a small positive exponent, a solution for c_2 of the congruence $(H + c_1)c_2 + (J + c_1H) \equiv 0 \pmod{q^h}$.

If $\gcd(H + c_1, q) = 1$, that is, if $H + c_1$ is not a multiple of q, then the solution is given by $\sigma \equiv -(J + c_1 H)(H + c_1)^{-1} \pmod{q^h}$. The inverse in the last congruence can be calculated by running the extended gcd algorithm (Algorithm 3.8) on $H + c_1$ and q^h. Then for each value of c_2 (in the range $c_1 \leqslant c_2 \leqslant M$) that is congruent to $\sigma \pmod{q^h}$, $\ln q$ is subtracted from the array location $\mathfrak{A}_{c_2}$.

If $q|(H + c_1)$, we find out $h_1 := \mathrm{v}_q(H + c_1) > 0$ and $h_2 := \mathrm{v}_q(J + c_1 H) \geqslant 0$. If $h_1 > h_2$, then for each value of c_2, the expression $T(c_1, c_2)$ is divisible by q^{h_2} and by no higher powers of q. So we subtract the quantity $h_2 \ln q$ from $\mathfrak{A}_{c_2}$ for all c_2. Finally, if $h_1 \leqslant h_2$, then we subtract $h_1 \ln q$ from $\mathfrak{A}_{c_2}$ for all c_2 and for $h > h_1$ solve the congruence as $\sigma \equiv -\left(\frac{J+c_1 H}{q^{h_1}}\right)\left(\frac{H+c_1}{q^{h_1}}\right)^{-1} \pmod{q^{h-h_1}}$.

Once the above procedure is carried out for each small prime q in the factor base and for each small exponent h, we check for which values of c_2, the value $\mathfrak{A}_{c_2}$ is equal (that is, sufficiently close) to 0. These are precisely the values of c_2 such that for the given c_1 the integer $T(c_1, c_2)$ factors smoothly over the small primes in the factor base.

As in the QSM for integer factorization, it is sufficient to have some approximate representations of the logarithms (like $\ln q$). *Incomplete sieving* and *large prime variation* can also be adopted as in the QSM.

Finally, we change c_1 and repeat the sieving process described above. It is easy to see that the sieving operations for all c_1 in the range $-M \leqslant c_1 \leqslant M$ take time $L[1]$ as announced earlier. Gaussian elimination involving sparse congruences in $L[1/2]$ variables also meets the same running time bound.

The second stage of the LSM can be performed in $L[1/2]$ time. Using a method similar to the second stage of the basic ICM leads to a huge running time ($L[3/2]$), because we have only $L[1/2]$ small primes in the factor base. We instead do the following. We start with a random j and try to obtain a factorization of the form $ag^j \equiv \left(\prod_q q^{v_q}\right)\left(\prod_u u^{w_u}\right) \pmod{p}$, where q runs over $L[1/2]$ small primes in the factor base and u runs over medium-sized primes, that is, primes less than $L[2]$. One can use an integer factorization algorithm to this effect. Lenstra's ECM is, in particular, recommended, since it can detect smooth integers fast. More specifically, about $L[1/4]$ random values of j need to be tried, before we expect to get an integer with the desired factorization. Each attempt of factorization using the ECM takes time less than $L[1/4]$.

Now, we have $\mathrm{ind}_g\, a \equiv -j + \sum_q v_q \,\mathrm{ind}_g\, q + \sum_u w_u \,\mathrm{ind}_g\, u \pmod{p-1}$. The indices $\mathrm{ind}_g\, q$ are available from the first stage, whereas for each u (with $w_u \neq 0$) the index $\mathrm{ind}_g\, u$ is calculated as follows. First we sieve in an interval of size $L[1/2]$ around $\sqrt{p}/u$ and collect integers y in this interval, which are smooth with respect to the $L[1/2]$ primes in the factor base. A second sieve in an interval of size $L[1/2]$ around H gives us a small integer c, such that $(H + c)yu - p$ is smooth again with respect to the $L[1/2]$ primes in the factor base. Since $H + c$ is in the factor base, we get $\mathrm{ind}_g\, u$. The reader can easily verify that computing individual logarithms $\mathrm{ind}_g\, a$ using this method takes time $L[1/2]$ as claimed earlier.

There are some other $L[1]$ methods (like the Gaussian integer method and the residue list sieve method) known for computing discrete logarithms in prime fields. We will not discuss these methods in this book. Interested readers may refer to Coppersmith et al. [59] to know about these $L[1]$ methods. A faster method (running time $L[0.816]$),

namely the cubic sieve method, is covered in Exercise 4.21. Now, we turn our attention to the best method known till date.

**** The number field sieve method**

The number field sieve method (NFSM) for solving the DLP in a prime field $\mathbb{F}_p$ is a direct adaptation of the NFSM used to factor integers (Section 4.3.4). As before, we let g be a generator of $\mathbb{F}_p^*$ and are interested in computing the index $d = \operatorname{ind}_g a$ for some $a \in \mathbb{F}_p^*$.

We choose an irreducible polynomial $f(X) \in \mathbb{Z}[X]$ with small integer coefficients and of degree d, and use the number field $K := \mathbb{Z}[X]/\langle f(X)\rangle \cong \mathbb{Q}(\alpha)$ for some root $\alpha \in \mathbb{C}$ of f. For the sake of simplicity, we consider the *special* case (SNFSM) that f is monic, $\mathfrak{O}_K$ is a PID, and $\mathfrak{O}_K = \mathbb{Z}[\alpha]$. We also choose an integer m such that $f(m) \equiv 0 \pmod p$ and define the ring homomorphism

$$\Phi : \mathfrak{O}_K = \mathbb{Z}[\alpha] \to \mathbb{F}_p, \quad \alpha \mapsto m \pmod p .$$

Finally, we predetermine a bound $\mathcal{B} \in \mathbb{N}$ and let $\mathcal{P}$ be the set of (rational) primes $\leqslant \mathcal{B}$, $\mathcal{Q}$ the set of prime ideals of $\mathfrak{O}_K$ of prime norms $\leqslant \mathcal{B}$, $\mathcal{H}$ a set of generators of the (principal) ideals $\mathfrak{q} \in \mathcal{Q}$ and $\mathcal{U}$ a set of generators of the group of units of $\mathfrak{O}_K$.

We try to find coprime integers c, d of small absolute values such that both $c + d\alpha$ and $\Phi(c + d\alpha) = c + dm$ are smooth with respect to $\mathcal{Q}$ and $\mathcal{P}$ respectively, that is, we have factorizations of the forms $c + dm = \prod_{p\in\mathcal{P}} p^{v_p}$ and $\langle c + d\alpha\rangle = \prod_{\mathfrak{q}\in\mathcal{Q}} \mathfrak{q}^{w_\mathfrak{q}}$ or equivalently, $c + d\alpha = \prod_{u\in\mathcal{U}} u^{r_u} \prod_{h\in\mathcal{H}} h^{w_h}$. But then $\Phi(c + d\alpha) = \prod_{u\in\mathcal{U}} \Phi(u)^{r_u} \prod_{h\in\mathcal{H}} \Phi(h)^{w_h} \equiv \prod_{p\in\mathcal{P}} p^{v_p} \pmod p$, that is,

$$\sum_{u\in\mathcal{U}} r_u \operatorname{ind}_g \Phi(u) + \sum_{h\in\mathcal{H}} w_h \operatorname{ind}_g \Phi(h) \equiv \sum_{p\in\mathcal{P}} v_p \operatorname{ind}_g p \pmod{p-1}. \qquad \textbf{(4.8)}$$

This motivates us to define the factor base as

$$B := \{\Phi(u) \mid u \in \mathcal{U}\} \cup \{\Phi(h) \mid h \in \mathcal{H}\} \cup \mathcal{P} .$$

We assume that $g \in \mathcal{P}$ so that we have the *free* relation $\operatorname{ind}_g g \equiv 1 \pmod{p-1}$.

Trying sufficiently many pairs (c, d) we generate many Relations (4.8). The resulting sparse linear system is solved for the unknown indices of the elements of B. This completes the first stage of the SNFSM.

In the second stage, we bring a to the scene in the following manner. First assume that a is small such that either a is $\mathcal{B}$-smooth, that is,

$$a = \prod_{p\in\mathcal{P}} p^{t_p},$$

or for some $\gamma \in \Phi^{-1}(a)$ the ideal $\langle\gamma\rangle = \gamma\mathfrak{O}_K$ can be written as a product of prime ideals of $\mathcal{Q}$, that is,

$$\langle\gamma\rangle = \prod_{\mathfrak{q}\in\mathcal{Q}} \mathfrak{q}^{t_\mathfrak{q}}$$

or, equivalently,

$$a = \Phi(\gamma) \equiv \prod_{u \in \mathcal{U}} \Phi(u)^{s_u} \prod_{h \in \mathcal{H}} \Phi(h)^{t_h} \pmod{p}.$$

In both the cases, taking logarithms and substituting the indices of the elements of the factor base (available from the first stage) yields $d = \operatorname{ind}_g a$.

However, a is not small, in general, and it is a non-trivial task to find a $\gamma \in \Phi^{-1}(a)$ such that $\langle \gamma \rangle$ is $\mathcal{Q}$-smooth. We instead write a as a product

$$a \equiv \prod_{i=1}^{r} a_i \pmod{p}, \tag{4.9}$$

where each a_i is small enough so that $\operatorname{ind}_g a_i$ can be computed using the method described above. This gives $\operatorname{ind}_g a \equiv \sum_{i=1}^{r} \operatorname{ind}_g a_i \pmod{p-1}$. In order to see how one can find a representation of a as a product of small integers as in Congruence (4.9), we refer the reader to Weber [300].

As in most variants of the ICM, the running time of the SNFSM is dominated by the first stage and under certain heuristic assumptions can be shown to be of the order of $L(p, 1/3, (32/9)^{1/3})$. Look at Section 4.3.4 to see how the different parameters can be set in order to achieve this running time. For the *general* NFS method (GNFSM), the running time is $L(p, 1/3, (64/9)^{1/3})$. The GNFSM has been implemented by Weber and Denny [301] for computing discrete logarithms modulo a particular prime having 129 decimal digits (see McCurley [189]).

4.4.4 Algorithms for Fields of Characteristic 2

We wish to compute the discrete logarithm $\operatorname{ind}_g a$ of an element $a \in \mathbb{F}_q^*$, $q = 2^n$, with respect to a primitive element g of $\mathbb{F}_q^*$. We work with the representation $\mathbb{F}_q \cong \mathbb{F}_2[X]/\langle f \rangle$ for some irreducible polynomial $f \in \mathbb{F}_2[X]$ with $\deg f = n$. For certain algorithms, we require f to be of special forms. This does not create enough difficulties, since it is easy to compute isomorphisms between two polynomial basis representations of $\mathbb{F}_q$ (Exercise 3.38).

Recall that we have defined the smoothness of an integer x in terms of the magnitudes of the prime divisors of x. Now, we deal with polynomials (over $\mathbb{F}_2$) and extend the definition of smoothness in the obvious way: that is, a polynomial is called smooth if it factors into irreducible polynomials of low degrees. The next theorem is an analog of Theorem 2.21 for polynomials. By an abuse of notation, we use $\psi(\cdot, \cdot)$ here also. The context should make it clear what we are talking about – smoothness of integers or of polynomials.

Theorem 4.1 Let $r, m \in \mathbb{N}$, $r^{1/100} \leqslant m \leqslant r^{99/100}$, and let $u := r/m$. Then the number of polynomials $f \in \mathbb{F}_2[X]$, $\deg f = r$, such that all irreducible factors of f have degrees $\leqslant m$, equals $2^r u^{-u+o(u)} = 2^r e^{-[(1+o(1))u \ln u]}$ as $u \to \infty$. In particular, the probability that the degrees of all irreducible factors of a randomly chosen polynomial in $\mathbb{F}_2[X]$ of degree r are $\leqslant m$ is asymptotically equal to

$$\psi(r, m) := u^{-u+o(u)} = e^{-[(1+o(1))u \ln u]}. \qquad \blacksquare$$

The above expression for $\psi(r,m)$, though valid asymptotically, gives good approximations for finite values of r and m. The condition $r^{1/100} \leqslant m \leqslant r^{99/100}$ is met in most practical situations. The probability $\psi(r,m)$ is a very sensitive function of $u = r/m$. For a fixed m, polynomials of smaller degrees have higher chances of being smooth (that is, of having all irreducible factors of degrees $\leqslant m$).

Now, let us consider the field $\mathbb{F}_q$ with $q = 2^n$. The elements of $\mathbb{F}_q$ are represented as polynomials of degrees $\leqslant n-1$. For a given m, the probability $\bar{\psi}(n,m)$ that a randomly chosen element of $\mathbb{F}_q$ has all irreducible factors of degrees $\leqslant m$ is then approximately given by $(\sum_{r=0}^{n-1} 2^r \psi(r,m))/2^n \approx \psi(n,m)\left(\frac{(ne/m)^{1/m}}{2-(ne/m)^{1/m}}\right) \rightarrow \psi(n,m)$, as $n, m \rightarrow \infty$ with $n^{1/100} \leqslant m \leqslant n^{99/100}$. We can, therefore, approximate $\bar{\psi}(n,m)$ by $\psi(n,m)$.

For many algorithms that we will come across shortly, we have $r \approx n/\alpha$ and $m \approx \sqrt{(n \ln n)/\beta}$ for some positive α and β, so that $u = r/m \approx \frac{\sqrt{\beta}}{\alpha}\sqrt{\frac{n}{\ln n}}$ and, consequently,

$$\psi(r,m) = e^{-[(1+o(1))\frac{\sqrt{\beta}}{2\alpha}\sqrt{n \ln n}]} = e^{-[(\frac{\sqrt{\beta}}{2\alpha\sqrt{\ln 2}}+o(1))\sqrt{\ln q \ln\ln q}]} = L\left[-\frac{\sqrt{\beta}}{2\alpha\sqrt{\ln 2}}\right].$$

The basic ICM

The idea of the basic ICM for $\mathbb{F}_{2^n}$ is analogous to that for prime fields. Now, the factor base B comprises all irreducible polynomials of $\mathbb{F}_2[X]$ having degrees $\leqslant m$. We choose $m \approx \sqrt{\frac{n \ln n}{2 \ln 2}}$. (As in the case of the basic ICM for prime fields, this can be shown to be the optimal choice.) By Approximation (2.5) on p 84, we then have $\#B \approx \frac{2^1}{1} + \frac{2^2}{2} + \cdots + \frac{2^m}{m} \approx \frac{2^{m+1}}{m} = L[1/\sqrt{2}]$.

In the first stage, we choose random α, $1 \leqslant \alpha \leqslant q-2$, compute g^α and check if g^α is B-smooth. If so, we get a relation. For a random α, the polynomial g^α is a random polynomial of degree $< n$ and hence has a probability of nearly $\psi(n,m) = L[-1/\sqrt{2}]$ of being smooth. Note that unlike integers a polynomial over $\mathbb{F}_2$ can be factored in probabilistic polynomial time (though for small m it may be preferable to do trial division by elements of B). Thus checking the smoothness of a random element of $\mathbb{F}_q^*$ can be done in (probabilistic) polynomial time, and each relation is available in expected time $L[1/\sqrt{2}]$. Since we need (slightly more than) $L[1/\sqrt{2}]$ relations for setting up the linear system, the relation collection stage runs in expected time $L[1/\sqrt{2}]L[1/\sqrt{2}] = L[\sqrt{2}]$. A sparse system with $L[1/\sqrt{2}]$ unknowns can also be solved in time $L[\sqrt{2}]$.

In the second stage, we need a single smooth polynomial of the form $g^\alpha a$. If α is randomly chosen, we expect to get this relation in time $L[1/\sqrt{2}]$. Therefore, the second stage is again faster than the first and the basic method takes a total expected running time of $L[\sqrt{2}]$. Recall that the basic method for $\mathbb{F}_p$ requires time $L[2]$. The difference arises because polynomial factorization is much *easier* than integer factorization.

We now explain a modification of the basic method, proposed by Blake et al. [23]. Let $h \in \mathbb{F}_q^*$: that is, a non-zero polynomial in $\mathbb{F}_2[X]$ of degree $< n$. If h is randomly chosen from $\mathbb{F}_q^*$ (as in the case of g^α or $g^\alpha a$ for random α), then we expect the degree of h to be close to n. Let us write $h \equiv h_1/h_2 \pmod f$ (f being the defining polynomial) with h_1 and h_2 each having degree $\approx n/2$. Then the ratio of the probability that *both* h_1 and h_2 are smooth to the probability that h is smooth is $\psi(n/2,m)^2/\psi(n,m) \approx 2^{n/m}$ (neglecting the $o(\,)$ terms). For practical values of n and m, this ratio of probabilities

can be substantially large implying that it is easier to get relations by trying to factor both h_1 and h_2 instead of trying to factor h. This is the key observation behind the modification due to Blake et al. [23]. Simple calculations show that this modification does not affect the asymptotic $L[\sqrt{2}]$ behaviour of the basic method, but it leads to considerable speed-up in practice.

In order to complete the description of the modification of Blake et al. [23], we mention an efficient way to write h as $h_1/h_2 \pmod f$. Since $0 \leqslant \deg h < n$ and since f is irreducible of degree n, we must have $\gcd(h, f) = 1$. During the iteration of the extended gcd algorithm we actually compute a sequence of polynomials u_k, v_k, x_k such that $u_k h + v_k f = x_k$ for all $k = 0, 1, 2, \ldots$. At the start of the algorithm we have $u_0 = 1$, $v_0 = 0$ and $x_0 = h$. As the algorithm proceeds, the sequence $\deg u_k$ changes non-decreasingly, whereas the sequence $\deg x_k$ changes non-increasingly and at the end of the extended gcd algorithm we have $x_k = 1$ and the desired Bézout relation $u_k h + v_k f = 1$ with $\deg u_k \leqslant n - 1$. Instead of proceeding till the end of the gcd loop, we stop at the value $k = k'$ for which $\deg x_{k'}$ is closest to $n/2$. We will then usually have $\deg u_{k'} \approx n/2$, so that taking $h_1 = x_{k'}$ and $h_2 = u_{k'}$ serves our purpose.

The concept of large prime variation is applicable for the basic ICM. Moreover, if trial divisions are used for smoothness tests, one can employ the early abort strategy. Despite all these modifications the basic variant continues to be rather slow. Our hunt for faster algorithms continues.

The adaptation of the linear sieve method

The LSM for prime fields can be readily adapted to the fields $\mathbb{F}_q$, $q = 2^n$. Let us assume that the defining polynomial f is of the special form $f(X) = X^n + f_1(X)$, where $\deg f_1$ is *small*. The total number of choices for such f with $\deg f_1 < k$ is 2^k. Under the assumption that irreducible polynomials (over $\mathbb{F}_2$) of degree n are randomly distributed among the set of polynomials of degree n, we expect to find an irreducible polynomial $f = X^n + f_1$ for $\deg f_1 = \mathrm{O}(\lg n)$ (see Approximation (2.5) on p 84). In particular, we may assume that $\deg f_1 \leqslant n/2$.

Let $k := \lceil n/2 \rceil$ and $\sigma := 2k - n \in \{0, 1\}$. For polynomials $h_1, h_2 \in \mathbb{F}_2[X]$ of small degrees, we then have

$$(X^k + h_1)(X^k + h_2) \equiv X^\sigma f_1 + (h_1 + h_2)X^k + h_1 h_2 \pmod f .$$

The right side of the congruence, namely,

$$T(h_1, h_2) := X^\sigma f_1 + (h_1 + h_2)X^k + h_1 h_2 ,$$

has degree slightly larger than $n/2$. This motivates the following algorithm.

We take $m \approx \sqrt{\frac{n \ln n}{4 \ln 2}}$ and let the factor base B be the (disjoint) union of B_1 and B_2, where B_1 contains irreducible polynomials of degrees $\leqslant m$, and where B_2 contains polynomials of the form $X^k + h$, $\deg h \leqslant m$. Both B_1 and B_2 (and hence B) contain $L[1/2]$ elements. For each $X^k + h_1, X^k + h_2 \in B_2$, we then check the smoothness of $T(h_1, h_2)$ over B_1. Since $\deg T(h_1, h_2) \approx n/2$, the probability of finding a smooth candidate per trial is $L[-1/2]$. Therefore, trying $L[1]$ values of the pair (h_1, h_2) is expected to give $L[1/2]$ relations (in $L[1/2]$ variables). Since factoring each $T(h_1, h_2)$ can be performed in probabilistic polynomial time, the relation collection stage takes

time $L[1]$. Gaussian elimination (with sparse congruences) can be done in the same time. As in the case of the LSM for prime fields, the second stage can be carried out in time $L[1/2]$. To sum up, the LSM for fields of characteristic 2 takes $L[1]$ running time.

Note that the running time $L[1]$ is achievable in this case without employing any sieving techniques. This is again because checking the smoothness of each $T(h_1, h_2)$ can simply be performed in polynomial time. Application of *polynomial sieving*, though unable to improve upon the $L[1]$ running time, often speeds up the method in practice. We will describe such a sieving procedure in connection with Coppersmith's algorithm that we describe next.

Coppersmith's algorithm

Coppersmith's algorithm is the fastest algorithm known to compute discrete logarithms in finite fields $\mathbb{F}_q = \mathbb{F}_{2^n}$ of characteristic 2. Theoretically it achieves the (heuristic) running time $L(q, 1/3, c)$ and is, therefore, *subexponentially* faster than the $L[c'] = L(q, 1/2, c')$ algorithms described so far. Gordon and McCurley have made aggressive attempts to compute discrete logarithms in fields as large as $\mathbb{F}_{2^{503}}$ using Coppersmith's algorithm in tandem with a polynomial sieving procedure and, thereby, established the practicality of the algorithm.

In the basic method, each trial during the search for relations involves checking the smoothness of a polynomial of degree nearly n. The modification due to Blake et al. [23] replaces this by checking the smoothness of two polynomials of degree $\approx n/2$. For the adaptation of the LSM, on the other hand, we check the smoothness of a single polynomial of degree $\approx n/2$. In Coppersmith's algorithm, each trial consists of checking the smoothness of two polynomials of degrees $\approx n^{2/3}$. This is the basic reason behind the improved performance of Coppersmith's algorithm.

To start with we make the assumption that the defining polynomial f of $\mathbb{F}_q$ is of the form $f(X) = X^n + f_1(X)$ with $\deg f_1 = \mathrm{O}(\lg n)$. We have argued earlier that an irreducible polynomial f of this special form is expected to be available. We now choose three integers m, M, k such that

$$m \approx \alpha n^{1/3}(\ln n)^{2/3},\ M \approx \beta n^{1/3}(\ln n)^{2/3} \text{ and } 2^k \approx \gamma n^{1/3}(\ln n)^{-1/3},$$

where the (positive real) constants α, β and γ are to be chosen appropriately to optimize the running time. The factor base B comprises irreducible polynomials (over $\mathbb{F}_2$) of degrees $\leqslant m$. Let

$$l := \lfloor n/2^k \rfloor + 1\,,$$

so that $l \approx (1/\gamma)n^{2/3}(\ln n)^{1/3}$. Choose relatively prime polynomials $u_1(X)$ and $u_2(X)$ (in $\mathbb{F}_2[X]$) of degrees $\leqslant M$ and let

$$h_1(X) := u_1(X)X^l + u_2(X) \text{ and } h_2(X) := (h_1(X))^{2^k} \bmod f(X).$$

But then, since $\mathrm{ind}_g\, h_2 \equiv 2^k \,\mathrm{ind}_g\, h_1 \pmod{q-1}$, we get a relation if both h_1 and h_2 are smooth over B. By choice, $\deg h_1$ is clearly $\tilde{\mathrm{O}}(n^{2/3})$, whereas

$$h_2(X) \equiv u_1(X^{2^k})X^{l2^k} + u_2(X^{2^k}) \equiv u_1(X^{2^k})X^{l2^k - n}f_1(X) + u_2(X^{2^k}) \pmod f$$

and, therefore, $\deg h_2 = \tilde{\mathrm{O}}(n^{2/3})$ too.

For each pair (u_1, u_2) of relatively prime polynomials of degrees $\leqslant M$, we compute h_1 and h_2 as above and collect all the relations corresponding to the smooth values of both h_1 and h_2. This gives us the desired (sparse) system of linear congruences in the unknown indices of the elements of B, which is subsequently solved modulo $q-1$.

The choice $\alpha = \beta = (\frac{2}{3\ln 2})^{2/3}$ and $\gamma = \alpha^{-1/2}$ gives the optimal running time of the first stage as

$$e^{[(2\alpha \ln 2)+o(1))n^{1/3}(\ln n)^{2/3}]} = L(q, 1/3, 2\alpha/(\ln 2)^{1/3}) \approx L(q, 1/3, 1.526).$$

The second stage of Coppersmith's algorithm is somewhat involved. The factor base now contains only nearly $L(q, 1/3, 0.763)$ elements. Therefore, finding a relation using a method similar to the second stage of the basic method requires time $L(q, 2/3, c)$ for some c, which is much worse than even $L[c'] = L(q, 1/2, c')$. To work around this difficulty we start by finding a polynomial $g^\alpha a$ all of whose irreducible factors have degrees $\leqslant n^{2/3}(\ln n)^{1/3}$. This takes time of the order of $L(q, 1/3, c_1)$ (where $c_1 \approx 0.377$) and gives us $g^\alpha a = \prod_i v_i$, where v_i have degrees $\leqslant n^{2/3}(\ln n)^{1/3}$. Note that the number of v_i is less than n, since $\deg(g^\alpha a) < n$. We then have

$$\operatorname{ind}_g a \equiv -\alpha + \sum_i \operatorname{ind}_g v_i \pmod{q-1}.$$

All these v_i need not belong to the factor base, so we cannot simply substitute the values of $\operatorname{ind}_g v_i$. We instead reduce the problem of computing each $\operatorname{ind}_g v_i$ to the problem of computing $\operatorname{ind}_g v_{ii'}$ for several i' with $\deg v_{ii'} \leqslant \sigma \deg v_i$ for some constant $0 < \sigma < 1$. Subsequently, computing each $\operatorname{ind}_g v_{ii'}$ is reduced to computing $\operatorname{ind}_g v_{ii'i''}$ for several i'' with $\deg v_{ii'i''} < \sigma \deg v_{ii'}$. Repeating this process, we eventually end up with the polynomials in the factor base. Because reduction of a polynomial generates new polynomials with degrees reduced by at least the constant factor σ, it is clear that the recursion depth is $O(\ln n)$. Now, if for each i the number of i' is $\leqslant n$ and for each i' the number of i'' is $\leqslant n$ and so on, we have to carry out the reduction of $\leqslant n^{O(\ln n)} = e^{O((\ln n)^2)} = L(q, 1/3, 0)$ polynomials. Therefore, if each reduction can be performed in time $L(q, 1/3, c_2)$, the second stage will run in time $L(q, 1/3, \max(c_1, c_2))$.

In order to explain how a polynomial v of degree $\leqslant d \leqslant n^{2/3}(\ln n)^{1/3}$ can be reduced in the desired time, we choose $k \in \mathbb{N}$ such that $2^k \approx \sqrt{n/d}$, and let $l := \lfloor n/2^k \rfloor + 1$. As in the first stage, we fix a suitable bound M, choose relatively prime polynomials $u_1(X), u_2(X)$ of degrees $\leqslant M$ and define

$$h_1(X) := u_1(X)X^l + u_2(X)$$

and

$$h_2(X) := (h_1(X))^{2^k} \operatorname{rem} f(X) = u_1(X^{2^k})X^{l2^k - n} f_1(X) + u_2(X^{2^k}).$$

The polynomials u_1 and u_2 should be so chosen that $v | h_1$. We see that h_1 and h_2 have *low* degrees and we try to factor h_1/v and h_2. Once we get a factorization of the form

$$h_2(X) = \prod_i v_i(X) \equiv h_1(X)^{2^k} \equiv \left(v(X) \prod_j w_j(X) \right)^{2^k} \pmod{f(X)}$$

with $\deg v_i, \deg w_j < \sigma \deg v$, we have the desired reduction of v, namely,

$$\sum_i \operatorname{ind}_g v_i \equiv 2^k (\operatorname{ind}_g v + \sum_j \operatorname{ind}_g w_j) \pmod{q-1},$$

that is, the reduction of computation of $\operatorname{ind}_g v$ to that of all $\operatorname{ind}_g v_i$ and $\operatorname{ind}_g w_j$. With the choice $M \approx (n^{1/3}(\ln n)^{2/3}(\ln 2)^{-1} + \deg v)/2$ and $\sigma = 0.9$, reduction of each polynomial can be shown to run in time $L(q, 1/3, (\ln 2)^{-1/3}) \approx L(q, 1/3, 1.130)$. Thus the second stage of Coppersmith's algorithm runs in time $L(q, 1/3, 1.130)$ and is faster than the first stage.

Large prime variation is a useful strategy to speed up Coppersmith's algorithm. In case of trial divisions for smoothness tests, early abort strategy can also be applied. However, a more efficient idea (though seemingly non-collaborative with the early abort strategy) is to use *polynomial sieving* as introduced by Gordon and McCurley.

Recall that in the first stage we take relatively prime polynomials u_1 and u_2 of degrees $\leqslant M$ and check the smoothness of both $h_1(X) = u_1(X)X^l + u_2(X)$ and $h_2(X) = h_1(X)^{2^k} \operatorname{rem} f(X)$. We now explain the (incomplete) sieving technique for filtering out the (non-)smooth values of $h_1 = (h_1)_{u_1,u_2}$ for the different values of u_1 and u_2. To start with we fix u_1 and let u_2 vary. We need an array $\mathfrak{A}$ indexed by u_2, a polynomial of degree $\leqslant M$. Clearly, u_2 can assume 2^{M+1} values and so $\mathfrak{A}$ must contain 2^{M+1} elements. To be very concrete we will denote by $\mathfrak{A}_{u_2(X)}$ the location $\mathfrak{A}_{u_2(2)}$, where $u_2(2) \geqslant 0$ is the integer obtained canonically by substituting 2 for X in $u_2(X)$ considered to be a polynomial in $\mathbb{Z}[X]$ with coefficients 0 and 1. We initialize all the locations of $\mathfrak{A}$ to zero.

Let $t = t(X)$ be a small irreducible polynomial in the factor base B (or a small power of such an irreducible polynomial) with $\delta := \deg t$. The values of u_2 for which t divides $(h_1)_{u_1,u_2}$ satisfy the polynomial congruence $u_2(X) \equiv u_1(X)X^l \pmod{t}$. Let u_2^* be the solution of this congruence with $\delta^* := \deg u_2^* < \deg t$. If $\delta^* > M$, then no value of u_2 corresponds to smooth $(h_1)_{u_1,u_2}$. So assume that $\delta^* \leqslant M$. If $\delta > M$, then the only value of u_2 for which $(h_1)_{u_1,u_2}$ is smooth is $u_2 = u_2^*$. So we may also assume that $\delta \leqslant M$. Then the values of u_2 that makes $(h_1)_{u_1,u_2}$ smooth are given by $u_2(X) = u_2^*(X) + v(X)t(X)$ for all polynomials $v(X)$ of degrees $\leqslant M - \delta$. For each of these $2^{M-\delta+1}$ values of u_2, we add $\delta = \deg t$ to the location $\mathfrak{A}_{u_2}$.

When the process mentioned in the last paragraph is completed for all $t \in B$, we find out for which values of u_2 the array locations $\mathfrak{A}_{u_2}$ contain values close to $\deg(h_1)_{u_1,u_2}$. These values of u_2 correspond to the smooth values of $(h_1)_{u_1,u_2}$ for the chosen u_1. Finally, we vary u_1 and repeat the sieving procedure again.

In each sieving process described above, we have to find out all the values $u_2 = u_2^* + vt$ as v runs through all polynomials of degrees $\leqslant M - \delta$. We may choose the different possibilities for v in any sequence, compute the products vt and then add these products to u_2^*. While doing so serves our purpose, it is not very efficient, because computing each u_2 involves performing a polynomial multiplication vt. Gordon and McCurley's trick steps through all the possibilities of v in a clever sequence that helps one get each value of u_2 from the previous one by a much reduced effort (compared to polynomial multiplication). The $2^{M-\delta+1}$ choices of v can be naturally mapped to the bit strings of length (exactly) $M - \delta + 1$ (with the coefficients of lower powers of X appearing later in the sequence). This motivates using the following concept.

Definition 4.2 Let $d \in \mathbb{N}$. Then the (*binary*) *gray code* $G_1^d, G_2^d, \ldots, G_{2^d}^d$ of *dimension* d is a sequence of all (that is, 2^d) bit strings of length d defined inductively as follows. For $d = 1$, we define $G_1^1 := 0$ and $G_2^1 := 1$, whereas for $d > 1$ we define

$$G_l^d := \begin{cases} 0G_l^{d-1} & \text{if } 1 \leqslant l \leqslant 2^{d-1}, \\ 1G_{2^d-l+1}^{d-1} & \text{if } 2^{d-1} < l \leqslant 2^d, \end{cases}$$

where juxtaposition denotes string concatenation. □

For example, the gray code of dimension 2 is $00, 01, 11, 10$ and that of dimension 3 is $000, 001, 011, 010, 110, 111, 101, 100$. Proposition 4.1 can be easily proved by induction on the dimension d.

Proposition 4.1 Let $d \in \mathbb{N}$ and let $G_1^d, G_2^d, \ldots, G_{2^d}^d$ be the gray code of dimension d. For any i, $1 \leqslant i < 2^d$, the bit strings G_i^d and G_{i+1}^d differ in exactly one bit position $b(i)$. This position is given by $b(i) = \mathrm{v}_2(i)$, where $\mathrm{v}_2(i)$ denotes the multiplicity of 2 in i. ■

Back to our sieving business! Let us agree to step through the values of v in the sequence $v_1, v_2, \ldots, v_{2^{M-\delta+1}}$, where v_i corresponds to the bit string $G_i^{M-\delta+1}$ for the $(M - \delta + 1)$-dimensional gray code. Let us also call the corresponding values of u_2 as $(u_2)_i = u_2^* + v_i t$. Now, v_1 is 0 and the corresponding $(u_2)_1 = u_2^*$ is available at the beginning. By Proposition 4.1 we have for $1 \leqslant i < 2^{M-\delta+1}$ the equality $v_{i+1} = v_i + X^{\mathrm{v}_2(i)}$, so that $(u_2)_{i+1} = (u_2)_i + X^{\mathrm{v}_2(i)} t$. Computing the product $X^{\mathrm{v}_2(i)} t$ involves shifting the coefficients of t and is done efficiently using bit operations only (assuming data structures introduced in Section 3.5). Thus $(u_2)_{i+1}$ is obtained from $(u_2)_i$ by a shift followed by a polynomial addition. This is much faster than computing $(u_2)_{i+1}$ directly as $u_2^* + v_{i+1} t$.

We mentioned earlier that efficient implementations of Coppersmith's algorithm allows one to compute, in feasible time, discrete logarithms in fields as large as $\mathbb{F}_{2^{503}}$. However, for much larger fields, say for $n \geqslant 1024$, this algorithm is still not a practical breakthrough. The intractability of the DLP continues to remain cryptographically exploitable.

EXERCISE SET 4.4

4.15 Binary search Let $\leqslant$ be a total order on a set S (finite or infinite) and let $a_1 \leqslant a_2 \leqslant \cdots \leqslant a_m$ be a given sequence of elements of S. Device an algorithm that, given an arbitrary element $a \in S$, determines using only $\mathrm{O}(\lg m)$ comparisons in S whether $a = a_i$ for some $i = 1, \ldots, m$ and, if so, returns i. **[H]**

4.16 (a) Show that any map $\varphi : \mathbb{F}_q \to \mathbb{F}_q$ can be represented uniquely as a polynomial of degree $< q$. **[H]**
(b) The set S of all maps $\mathbb{F}_q \to \mathbb{F}_q$ is a ring under point-wise addition and multiplication. Prove the ring isomorphism $S \cong \mathbb{F}_q[X]/\langle X^q - X \rangle$.

4.17 Let p be a prime and g a primitive element of $\mathbb{F}_p^*$. For $a \in \mathbb{F}_p^*$, prove the explicit formula $\mathrm{ind}_g\, a \equiv \sum_{j=1}^{p-2} \frac{a^j}{1-g^j} \pmod p$. What is the problem in using this formula for computing indices in $\mathbb{F}_p^*$?

4.18 In the basic ICM for the prime field $\mathbb{F}_p$, we try to factor random powers g^α over the factor base $B = \{q_1, \ldots, q_t\}$. In addition to the canonical representative of g^α in the set $\{1, \ldots, p-1\}$, one can also check for the smoothness of the integers $g^\alpha + kp$ for $-M \leqslant k \leqslant M$, where M is a small positive integer (to be determined experimentally).
(a) Let $\rho_{k,i} := (g^\alpha + kp) \text{ rem } q_i$ for $i = 1, \ldots, t$ and for $-M \leqslant k \leqslant M$. How can one compute these remainders $\rho_{k,i}$ efficiently? Device an algorithm that checks the smoothness of all $g^\alpha + kp$ using the values $\rho_{k,i}$. **[H]**
(b) Device an algorithm that uses a sieve over the interval $-M \leqslant k \leqslant M$.
(c) Explain how the above two strategies can be modified to work for the field $\mathbb{F}_{2^n}$.

4.19 **(a)** Show that for the LSM over $\mathbb{F}_p$ the average $\bar{T}$ and the maximum $T_{\max}$ of $|T(c_1, c_2)|$ over all values of c_1, c_2 (that is, for $-M \leqslant c_1 \leqslant c_2 \leqslant M$) are approximately $\frac{2}{3}HM$ and $2HM$, respectively. **[H]**
(b) For real $0 \leqslant \eta \leqslant 1$, let $\mathfrak{T}(\eta) := \#\{(c_1, c_2) \mid -M \leqslant c_1 \leqslant c_2 \leqslant M, |T(c_1, c_2)| \leqslant \eta T_{\max}\}$ and let $\mathfrak{t}(\eta) := \mathfrak{T}(\eta)/\mathfrak{T}(1)$. Show that $\mathfrak{t}(\eta) \approx \eta(2 - \eta)$. (This shows that the distribution of $T(c_1, c_2)$ is not really random.)

4.20 Consider the following modification of the LSM for $\mathbb{F}_p$. Define for $r \in \mathbb{N}$ the integers $H_r := \lfloor \sqrt{rp} \rfloor + 1$ and $J_r := H_r^2 - rp$. Choose a small $s \in \mathbb{N}$ and repeat the linear sieve method for each r, $1 \leqslant r \leqslant s$, that is, check the smoothness (over the first $t = L[1]$ primes) of the integers $T_r(c_1, c_2) := J_r + (c_1 + c_2)H_r + c_1c_2$ for all $1 \leqslant r \leqslant s$, $-\mu \leqslant c_1 \leqslant c_2 \leqslant \mu$. Let $\bar{T}'$ be the average of $|T_r(c_1, c_2)|$ over all choices of r, c_1 and c_2. Show that $\bar{T}'/\bar{T} \approx \frac{\mu}{M} \frac{\sqrt{1} + \sqrt{2} + \cdots + \sqrt{s}}{s}$, where $\bar{T}$ is as defined in Exercise 4.19. In particular, $\bar{T}' < \bar{T}$ for both the choices: (1) $\mu = \lfloor M/\sqrt{s} \rfloor$ and (2) $\mu = \lfloor M/s \rfloor$, that is, on an average we check smaller integers for smoothness under this modified strategy. Determine the size of the factor base and the total number of integers $T_r(c_1, c_2)$ checked for smoothness for the two values of μ given above.

4.21 **Cubic sieve method (CSM) for $\mathbb{F}_p$** Let the integers x, y, z satisfy $x^3 \equiv y^2 z \pmod{p}$ with $x^3 \neq y^2 z$. Assume that each of x, y, z is $\mathrm{O}(p^\xi)$.
(a) Show that for integers a, b, c with $a + b + c = 0$ one has

$$(x + ay)(x + by)(x + cy) \equiv y^2 T(a, b, c) \pmod{p},$$

where $T(a, b, c) := z + (ab + ac + bc)x + (abc)y = -b(b + c)(x + cy) + (z - c^2 x)$. Since x, y, z are $\mathrm{O}(p^\xi)$, we have $T(a, b, c) = \mathrm{O}(p^\xi)$ for small values of a, b, c.
(b) For the CSM, the factor base B comprises all primes $q_1, \ldots, q_t$ with $t = L[\sqrt{\xi/2}]$ together with the integers $x + ay$, $-M \leqslant a \leqslant M$, $M = L[\sqrt{\xi/2}]$. If $T(a, b, c)$ factors completely over $q_1, \ldots, q_t$, we get a relation. Show that if we check the smoothness of $T(a, b, c)$ for all $-M \leqslant a \leqslant b \leqslant c \leqslant M$ with $a + b + c = 0$, we expect to get enough relations to compute the discrete logarithms of elements of B.
(c) In order to carry out sieving, fix c and let b vary. Specify the details of the sieving process. **[H]**
(d) Specify an algorithm for the second stage of the CSM. **[H]**
(e) Show that the expected running time of the CSM is $L[\sqrt{2\xi}]$. Therefore, if $\xi < 1/2$, the CSM is asymptotically faster than the LSM method, since the LSM runs in time $L[1]$. The best possible value $\xi = 1/3$ corresponds to a running time $L[\sqrt{2/3}] \approx L[0.816]$ of the CSM.

4.22 The problem with the CSM is that it is not known how to efficiently compute a solution of the congruence

$$x^3 \equiv y^2 z \pmod{p} \tag{4.10}$$

subject to the condition that $x^3 \neq y^2 z$ and $x, y, z = \mathrm{O}(p^\xi)$ for $1/3 \leqslant \xi < 1/2$. In this exercise, we estimate the number of solutions of Congruence (4.10).
(a) Show that the total number of solutions of Congruence (4.10) modulo p with $x, y, z \in \mathbb{Z}_p^*$ is $(p-1)^2$ which is $\Theta(p^2)$.
(b) Show that the total number of solutions of Congruence (4.10) modulo p with $x, y, z \in \mathbb{Z}_p^*$ and $x^3 \neq y^2 z$ is also $\Theta(p^2)$.
(c) Under the heuristic assumption that the solutions (x, y, z) of Congruence (4.10) are randomly distributed in $\mathbb{Z}_p^3$, deduce that the expected number of solutions of Congruence (4.10) modulo p with $x, y, z \in \mathbb{Z}_p^*$, $x^3 \neq y^2 z$, and $1 \leqslant x, y, z \leqslant p^\xi$, $1/3 \leqslant \xi \leqslant 1$, is nearly $p^{3\xi-1}$. (Therefore, if ξ is slightly larger than $1/3$, we expect to get a solution. It is not known how to compute such a solution in polynomial (or even subexponential) time. However, for certain values of p a solution is *naturally* available, for example, if p (or a small multiple of p) is close to an integer cube.)

4.23 Adaptation of CSM for $\mathbb{F}_{2^n}$ Let $\mathbb{F}_{2^n}$ be represented as $\mathbb{F}_2[X]/\langle f \rangle$, where the defining polynomial f is of the form $f(X) = X^n + f_1(X)$ with $\deg f_1 \leqslant n/3$. Let $k := \lceil n/3 \rceil$. Show that for polynomials $h_1, h_2 \in \mathbb{F}_2[X]$ of small degrees $(X^k + h_1(X))(X^k + h_2(X))(X^k + h_1(X) + h_2(X)) \bmod f(X)$ is of degree slightly larger than $n/3$. Device an ICM for solving the DLP in $\mathbb{F}_{2^n}$ based on this observation. What is the best running time of this method? **[H]**

* 4.5 The Elliptic Curve Discrete Logarithm Problem (ECDLP)

Unlike the finite field DLP, there are no general-purpose subexponential algorithms to solve the ECDLP. Though good algorithms are known for certain specific types of elliptic curves, all known algorithms that apply to general curves take fully exponential time. The square root methods of Section 4.4 are the fastest known methods for solving the ECDLP over an arbitrary curve. As a result, elliptic curves are gaining popularity for building cryptosystems. The absence of subexponential algorithms implies that smaller fields can be chosen compared to those needed for cryptosystems based on the (finite field) DLP. This, in particular, results in smaller sizes of keys.

We start with Menezes, Okamoto and Vanstone's (*MOV*) algorithm that reduces the ECDLP in a curve over $\mathbb{F}_q$ to the DLP over the field $\mathbb{F}_{q^k}$ for some suitable $k \in \mathbb{N}$. Since, the DLP can be solved in subexponential time, the ECDLP is also solved in that time, provided that the extension degree $k = [\mathbb{F}_{q^k} : \mathbb{F}_q]$ is small. For supersingular curves, one can choose $k \leqslant 6$. For non-supersingular curves, this k is quite large, in general, and the MOV reduction takes exponential time.

A linear-time algorithm is known to solve the ECDLP over anomalous curves (that is, curves with trace of Frobenius equal to 1). This algorithm is called the *SmartASS method* after its inventors Smart, Araki, Satoh and Semaev [257, 265, 282].

J. H. Silverman [277] has proposed an algorithm known as the *xedni calculus method* for solving the ECDLP over an arbitrary curve. Rigorous running times of this algorithm are not known, however heuristic analysis and experiments suggest that this algorithm is not really practical.

Let E be an elliptic curve over a finite field $\mathbb{F}_q$ and let $P \in E(\mathbb{F}_q)$ be of order m. We want to compute $\mathrm{ind}_P Q$ (if it exists) for a point $Q \in E(\mathbb{F}_q)$. Unless it is necessary, we will not assume any specific defining equation for E or a specific value of q.

** 4.5.1 The MOV Reduction

Let us first look at the structure of the group $E_{\bar{K}}[m]$ of m-torsion points on an elliptic curve defined over K. Here $\bar{K}$ is the algebraic closure of K.

Theorem 4.2 Let K be a field of characteristic $p \in \mathbb{N}_0$, $m \in \mathbb{N}$ and E an elliptic curve defined over K. We consider two separate cases:[5]

(1) If $p = 0$ or if $p > 0$ does not divide m, then $E_{\bar{K}}[m] \cong \mathbb{Z}_m \oplus \mathbb{Z}_m$. In particular, $\#E_{\bar{K}}[m] = m^2$ in this case.

(2) If $p > 0$, then either $E_{\bar{K}}[p^e] = \{\mathcal{O}\}$ for all $e \in \mathbb{N}$ or $E_{\bar{K}}[p^e] \cong \mathbb{Z}_{p^e}$ for all $e \in \mathbb{N}$. ■

Now, let E be an elliptic curve defined over a finite field K of characteristic p. Let $m \in \mathbb{N}$ with $\gcd(m, p) = 1$. We use the shorthand notation $E[m]$ for $E_{\bar{K}}[m]$ (and *not* for $E_K[m]$). We want to define a function

$$e_m : E[m] \times E[m] \to \mu_m,$$

where $\mu_m \subseteq \bar{K}$ is the group of m-th roots of unity (Exercise 4.24). This function e_m, known as the *Weil pairing*, helps us reduce the ECDLP in $E(\mathbb{F}_q)$ to the DLP in a suitable field $\mathbb{F}_{q^k}$. Let $P, R \in E[m]$. The definition of $e_m(P, R)$ calls for using divisors on E. Recall from Exercise 2.125 that a divisor $\sum_{T \in E(\bar{K})} m_T[T]$ belongs to $\mathrm{Prin}_{\bar{K}}(E)$ (that is, is the divisor of a rational function on E) if and only if $\sum_{T \in E(\bar{K})} m_T = 0$ and $\sum_{T \in E(\bar{K})} m_T T = \mathcal{O}$. Since $mR = \mathcal{O}$, there is a rational function $f \in \bar{K}(E)$ such that $\mathrm{Div}(f) = m[R] - m[\mathcal{O}]$. Now, $R \in E[m^2]$ as well and $p \not| m^2$. Hence, by Theorem 4.2 there exists a point R' of order m^2 such that $R = mR'$. Since, $\#E[m] = m^2$, it follows that $\mathcal{O} = m^2 R' = \sum_{T \in E[m]} ((R' + T) - T)$ and, therefore, there exists a rational function $g \in \bar{K}(E)$ with $\mathrm{Div}(g) = \sum_{T \in E[m]} ([R' + T] - [T])$. The functions f and g as introduced above are unique up to multiplication by elements of $\bar{K}^*$. One can show that we can choose f and g in such a manner that $f \circ \lambda_m = g^m$, where $\lambda_m : E(\bar{K}) \to E(\bar{K})$ is the multiplication map $Q \mapsto mQ$. Then for $P \in E[m]$ and $U \in E(\bar{K})$ we have $g^m(P + U) = f(mP + mU) = f(mU) = g^m(U)$. Since g has only finitely many poles and zeros (whereas $E(\bar{K})$ is infinite), we can choose U such that both $g(U)$ and $g(P + U)$ are defined and non-zero. For such a point U, we then have $g(P + U)/g(U) \in \mu_m$ and define

$$e_m(P, R) := g(P + U)/g(U)\,.$$

[5] For the MOV reduction, only the first case is important.

The right side can be shown to be independent of the choice of U. The relevant properties of the Weil pairing e_m are now listed.

Proposition 4.2 Let $P, P', R, R' \in E[m]$ and $a, b \in \mathbb{Z}$. Then we have:

Identity	$e_m(P,P)$	$= 1.$
Alternation	$e_m(P,R)$	$= e_m(R,P)^{-1}.$
Bilinearity	$e_m(P+P',R)$	$= e_m(P,R)e_m(P',R),$
	$e_m(P,R+R')$	$= e_m(P,R)e_m(P,R'),$
	$e_m(aP,bR)$	$= (e_m(P,R))^{ab}.$
Non-degeneracy	$e_m(P,\mathcal{O})$	$= 1.$
	If $e_m(P,T) = 1$ for all $T \in E[m]$, then $P = \mathcal{O}$.	■

The above definition of e_m is not computationally effective. We will see later how we can compute $e_m(P,T)$ in probabilistic polynomial time using an alternative (but equivalent) definition.

Algorithm 4.7 shows how the MOV reduction algorithm makes use of Weil pairing. We now clarify the subtle details of this algorithm.

Algorithm 4.7 **MOV reduction**

Input: A point $P \in E(\mathbb{F}_q)$ of order m, $\gcd(m,q) = 1$, and a multiple Q of P.
Output: The index $\operatorname{ind}_P Q$, that is, an integer l with $Q = lP$.
Steps:

```
Choose the smallest k ∈ ℕ such that E[m] ⊆ E(F_{q^k}).
while (1) {
  Choose a random point R ∈ E[m].
  α := e_m(P,R),  β := e_m(Q,R).   /* α,β ∈ μ_m ⊆ F*_{q^k} */
  l := ind_α β.   /* Discrete logarithm in F_{q^k} */
  if (Q = lP) { Return l. }
}
```

The correctness of the algorithm

From the bilinearity of the Weil pairing, it follows that if $Q = lP$, $0 \leqslant l < m$, then $\beta = e_m(Q,R) = e_m(lP,R) = e_m(P,R)^l = \alpha^l$. Thus treating $\operatorname{ind}_\alpha \beta$ as the least non-negative integer modulo $\operatorname{ord}\alpha$ we conclude that $l = \operatorname{ind}_\alpha \beta$ if and only if $\operatorname{ord}\alpha = m$, that is, α is a *primitive* m-th root of unity. That α is an m-th root of unity for any $R \in E[m]$ is obvious from the definition of e_m. We now show that there exists some $R \in E[m]$ for which $\alpha = e_m(P,R)$ is primitive.

Lemma 4.1 Let $P \in E[m]$ be of order m (so that P generates the subgroup $\langle P \rangle$ of order m in $E[m]$). Then for any $R_1, R_2 \in E[m]$ the cosets $R_1 + \langle P \rangle$ and $R_2 + \langle P \rangle$ are equal if and only if $e_m(P,R_1) = e_m(P,R_2)$.

Proof If $R_1 + \langle P \rangle = R_2 + \langle P \rangle$, then $R_1 = R_2 + rP$ for some integer r and so by bilinearity and identity of Weil pairing $e_m(P,R_1) = e_m(P,R_2)e_m(P,P)^r = e_m(P,R_2)$.

Conversely, let $e_m(P, R_1) = e_m(P, R_2)$. By Theorem 4.2, $E[m] \cong \mathbb{Z}_m \oplus \mathbb{Z}_m$ is generated by two elements of order m. We can take one of these elements to be P, let P' be the other element and write $R_1 - R_2 = aP + a'P'$ for some $a, a' \in \mathbb{Z}$. Then $e_m(P, R_1) = e_m(P, R_2 + aP + a'P') = e_m(P, R_2)e_m(P, P)^a e_m(P, a'P')$, whence it follows that $e_m(P, a'P') = 1$. Finally, for an arbitrary $T = bP + b'P' \in E[m]$, $b, b' \in \mathbb{Z}$, we have $e_m(a'P', T) = e_m(a'P', bP + b'P') = e_m(a'P', P)^b e_m(P', P')^{a'b'} = e_m(P, a'P')^{-b} = 1$. By the non-degeneracy property of e_m, it then follows that $a'P' = \mathcal{O}$, that is, $R_1 - R_2 = aP \in \langle P \rangle$. ■

As an immediate corollary to Lemma 4.1, the desired result follows.

Proposition 4.3 Let $P \in E[m]$ be of order m and let

$$S := \{R \in E[m] \mid e_m(P, R) \text{ is a primitive } m\text{-th root of unity}\}.$$

Then $\#S/\#E[m] = \phi(m)/m$. In particular, S is non-empty.

Proof There are m distinct cosets of $\langle P \rangle$ in $E[m]$. Now, as R ranges over all points of $E[m]$, the coset $R + \langle P \rangle$ ranges over all of these m possibilities and, accordingly by Lemma 4.1 the value $e_m(P, R)$ ranges over m distinct values. Since μ_m is cyclic of order m and hence with $\phi(m)$ generators, the theorem follows. ■

By Theorem 3.1, one should try an expected number of $\mathrm{O}(\ln \ln m)$ random points $R \in E[m]$ before a primitive m-th root $\alpha = e_m(P, R)$ is found.

Choosing k

Since $E[m]$ consists of finitely many (m^2) points, it is obvious that there exist finite values of k such that $E[m] \subseteq \mathbb{F}_{q^k}^2$. It can also be shown that if $E[m] \subseteq \mathbb{F}_{q^k}^2$, then $\mu_m \subseteq \mathbb{F}_{q^k}$ that is, $e_m(P, R) \in \mathbb{F}_{q^k}$ for all $P, R \in E[m]$. The computation of the discrete logarithm $\mathrm{ind}_\alpha \beta$ is then carried out in $\mathbb{F}_{q^k}$. For Algorithm 4.7 to be efficient, one requires k to be rather small. However, for most curves, k is rather large implying that the MOV reduction is impractical for these curves. For the specific class of curves, the so-called *supersingular curves*, one can choose k to be rather small, namely $k \leqslant 6$. We don't go to the details of the choices for k for various cases of supersingular curves, but refer the reader to Menezes [192].

Computing $e_m(P, R)$

We start with an alternative definition of the Weil pairing $e_m(P, R) \in \mu_m$ for $P, R \in E[m]$. First note that if $D = \sum_{T \in E(\bar{K})} m_T[T]$ is a divisor and if $f \in \bar{K}(E)$ is a rational function on E such that for every pole or zero T of f one has $m_T = 0$ (that is, such that $\mathrm{Div}(f)$ and T have disjoint supports), then one can define

$$f(D) := \sum_{T \in \mathrm{Supp}\, D} f(T)^{m_T}.$$

Choose points $U, V \in E(\bar{K})$ (where $K = \mathbb{F}_q$) and consider the divisors $D_P := [P + U] - [U]$ and $D_R := [R + V] - [V]$. Since $E(\bar{K})$ is infinite, one can choose both $P + U$

and U distinct from $R+V$ and V. Since $P, R \in E[m]$, it follows that mD_P and mD_R are principal, namely, there are rational functions f_P and f_R such that $\mathrm{Div}(f_P) = mD_P = m[P+U] - m[U]$ and $\mathrm{Div}(f_R) = mD_R = m[R+V] - m[V]$. One can show that

$$e_m(P,R) = f_P(D_R)/f_R(D_P), \tag{4.11}$$

independent of the choice of U and V as long as $f_P(D_R)$ and $f_R(D_P)$ are defined. Therefore, $e_m(P,R)$ can be computed efficiently, if f_P and f_R can be computed efficiently. To this effect we now describe an algorithm for computing the rational function f of a principal divisor $D = \mathrm{Div}(f) = \sum_{i=1}^{r} m_i[T_i]$, where $T_1, \ldots, T_r \in E(\bar{K})$. Since $\deg D = \sum_{i=1}^{r} m_i = 0$, we can write $D = \sum_{i=1}^{r} m_i([T_i] - [\mathcal{O}]) = \sum_{i=1}^{r} m_i([T_i] - [\mathcal{O}] + \mathrm{Div}(1))$. Suppose that we have an Algorithm A that, for a pair of reduced divisors

$$D_1 = [P_1] - [\mathcal{O}] + \mathrm{Div}(f_1)$$

and

$$D_2 = [P_2] - [\mathcal{O}] + \mathrm{Div}(f_2),$$

computes the sum (a reduced divisor)

$$D_3 = D_1 + D_2 = [P_3] - [\mathcal{O}] + \mathrm{Div}(f_3).$$

Then, f can be computed by repeated application of Algorithm A as follows.

(1) Compute for each $i = 1, \ldots, r$ the reduced divisor $\Delta_i = [Q_i] - [\mathcal{O}] + \mathrm{Div}(g_i) = m_i([T_i] - [\mathcal{O}] + \mathrm{Div}(1))$. Let $1 = a_{i1}, a_{i2}, \ldots, a_{it_i} = |m_i|$ be an addition chain for $|m_i|$ (Exercise 3.18). Clearly, $t_i - 1$ applications of Algorithm A computes Δ_i. Since we can choose $t_i \leqslant 2\lceil \lg |m_i| \rceil$, each Δ_i can be computed using $\mathrm{O}(\log |m_i|)$ applications of Algorithm A.

(2) Compute f by computing $D = \mathrm{Div}(f) = \Delta_1 + \cdots + \Delta_r$. This can be done by applying Algorithm A a total of $r-1$ times.

What remains is the description of Algorithm A that computes P_3 and f_3 from a knowledge of P_1, P_2, f_1 and f_2. Clearly, if $P_1 = \mathcal{O}$, then we have $P_3 = P_2$ and $f_3 = f_1 f_2$. Similar is the case for $P_2 = \mathcal{O}$. So assume $P_1 \neq \mathcal{O}$ and $P_2 \neq \mathcal{O}$. Let l_1 be the line passing through P_1 and P_2 and $P' := -(P_1 + P_2)$. First, assume that $P' \neq \mathcal{O}$. By Exercise 2.125, we have $\mathrm{Div}(l_1) = [P_1] + [P_2] + [P'] - 3[\mathcal{O}]$. Let l_2 be the (vertical) line passing through P' and $-P'$. Again by Exercise 2.125, we have $\mathrm{Div}(l_2) = [P'] + [-P'] - 2[\mathcal{O}]$. But then $D_3 = [P_1] + [P_2] - 2[\mathcal{O}] + \mathrm{Div}(f_1 f_2) = -[P'] + [\mathcal{O}] + \mathrm{Div}(f_1 f_2 l_1) = [-P'] - [\mathcal{O}] + \mathrm{Div}(f_1 f_2 l_1 / l_2)$, that is, we take $P_3 = -P' = P_1 + P_2$ and $f_3 = f_1 f_2 l_1 / l_2$. Finally, if $P' = \mathcal{O}$, then $\mathrm{Div}(l_1) = [P_1] + [P_2] - 2[\mathcal{O}]$ and, therefore, $D_3 = [P_1] + [P_2] - 2[\mathcal{O}] + \mathrm{Div}(f_1 f_2) = \mathrm{Div}(f_1 f_2 l_1) = [\mathcal{O}] - [\mathcal{O}] + \mathrm{Div}(f_1 f_2 l_1)$. Thus, in this case too, we take $P_3 = \mathcal{O} = P_1 + P_2$ and $f_3 = f_1 f_2 l_1 / l_2$ with $l_2 := 1$.

Before we finish the description of the MOV reduction, some comments are in order. First note that if $f_1, f_2 \in K(E)$ and $P_1, P_2 \in E(K)$, then both l_1 and l_2 are in $K(E)$ and the computation of f_3 and P_3 can be carried out by working in K only.

Second, consider the (general) case $P_3 \neq P_1, P_2, -P_3, \mathcal{O}$. Since $\mathrm{Div}(f_3) = [P_1] + [P_2] - [P_3] - [\mathcal{O}]$, the rational function f_3 has poles and is, therefore, undefined

only at the points P_3 and $\mathcal{O}$. f_3 is certainly defined at $-P_3$, but $l_2(-P_3) = 0$ and, therefore, evaluating $f_3(-P_3)$ as $(f_1 f_2 l_1)(-P_3)/l_2(-P_3)$ fails. Of course, there is a rational function g such that both $f_1 f_2 l_1 g$ and $l_2 g$ are defined and non-zero at $-P_3$, but finding such a rational function is an added headache. So we choose to continue to have the representation $f_3 = f_1 f_2 l_1 / l_2$ and agree *not* to evaluate f_3 at $-P_3$. Recall from Equation (4.11) that we want to evaluate f_P at D_R (that is, at $R + V$ and V) and also f_R and D_P (that is, at $P + U$ and U). Let us assume that we use the addition chain $1 = a_1, a_2, \ldots, a_t = m$ for m. This means that we cannot evaluate f_P at the points $\pm a_i(P + U)$ and $\pm a_i U$ for all $i = 1, \ldots, t$. Therefore, V should be chosen such that both $R + V$ and V are not one of these points. Similar constraints dictate the choice of U. However, if m is sufficiently large ($m \geqslant 1024$) and if we choose an addition chain of length $t \leqslant 2\lceil \lg m \rceil$, then it can be easily seen that for a random choice of (U, V) the evaluation of $f_P(D_R)$ or $f_R(D_P)$ fails with a probability of no more than $1/2$. Therefore, few random choices of (U, V) are expected to make the algorithm work. This is the only place where a probabilistic behaviour of the algorithm creeps in. In practice, however, this is not a serious problem, since we have much larger values of m (than 1024) and accordingly the above probability of failure becomes negligibly small.

Finally, note that if we multiply the factors f_1, f_2 and l_1 in the numerator, then the coefficients of the numerator grow very rapidly, when the algorithm is applied repeatedly. Thus we prefer to keep the numerator in the factored form. The same applies to the denominator as well.

** 4.5.2 The SmartASS Method

The *SmartASS method*, named after its inventors Smart[282], Satoh and Araki [257] and Semaev [265], is also called the *anomalous attack* to solve the ECDLP, since it is applicable to anomalous elliptic curves. Let $\mathbb{F}_p$ be a finite field of odd prime cardinality p and E an elliptic curve defined over $\mathbb{F}_p$. We assume that E is anomalous: that is, the trace of Frobenius of E at p is 1; that is, $\#E(\mathbb{F}_p) = p$. Since p is prime, the group $E(\mathbb{F}_p)$ is cyclic and, in particular, isomorphic to the additive group $(\mathbb{F}_p, +)$. This isomorphism is effectively exploited by the SmartASS method to give a polynomial time algorithm for computing ECDLP in the group $E(\mathbb{F}_p)$.

Before proceeding further we introduce some auxiliary results. Recall (Exercise 2.133) that a local PID is called a *discrete valuation ring* (*DVR*). Now, we see an equivalent definition of a DVR, that gives a justification to its name.

Definition 4.3 A *discrete valuation* on a field K is a surjective group homomorphism

$$v : (K^*, \cdot) \to (\mathbb{Z}, +)$$

such that for every $a, b \in K^*$ we have $v(a + b) \geqslant \min(v(a), v(b))$. We extend the definition of v to a map $K \to \mathbb{Z}$ by setting $v(0) = +\infty$. The set

$$\{a \in K \mid v(a) \geqslant 0\}$$

is a ring called the *valuation ring* of v. □

A DVR can be characterized as follows:

Proposition 4.4 Let R be an integral domain and let $K := \mathrm{Q}(R)$ be the field of fractions of R. Then R is a DVR if and only if there exists a discrete valuation $v : K \to \mathbb{Z}$ of K such that R is the valuation ring of v.

Proof [if] By definition, $R = \{a \in K \mid v(a) \geqslant 0\}$. We have $v(1) = v(1 \cdot 1) = v(1) + v(1)$, so that $v(1) = 0$. If $ab = 1$ for some $a, b \in R$, then $0 = v(1) = v(ab) = v(a) + v(b)$. Since $v(a), v(b) \geqslant 0$, it follows that $v(a) = v(b) = 0$. Conversely, let $v(a) = 0$ for some $a \in R$, $a \neq 0$. Now, $a^{-1} \in K$ and we have $0 = v(1) = v(aa^{-1}) = v(a) + v(a^{-1}) = v(a^{-1})$: that is, $a^{-1} \in R$. We conclude that $a \in R$ is a unit if and only if $v(a) = 0$. Any proper ideal of R consists only of non-units and hence is contained in the set $\mathfrak{m} := \{a \in R \mid v(a) > 0\}$ which is easily seen to be an ideal of R. Thus R is a local domain with maximal ideal $\mathfrak{m}$.

Let $i \in \mathbb{N}_0$ and define $\mathfrak{a}_i := \{a \in R \mid v(a) \geqslant i\}$. Clearly, each $\mathfrak{a}_i$ is an ideal of R. For an arbitrary non-zero ideal $\mathfrak{a}$ of R, consider $i := \min\{v(a) \mid a \in \mathfrak{a}\}$. If $i = 0$, then $\mathfrak{a}$ contains a unit, that is, $\mathfrak{a} = \langle 1 \rangle = \mathfrak{a}_0$. So assume $i > 0$. Clearly, $\mathfrak{a} \subseteq \mathfrak{a}_i$. Conversely, let $a \in \mathfrak{a}_i$, so that $v(a) \geqslant i$. Choose $b \in \mathfrak{a}$ with $v(b) = i$. But then $i \leqslant v(a) = v(ab^{-1}) + v(b) = v(ab^{-1}) + i$: that is, $v(ab^{-1}) \geqslant 0$; that is, $ab^{-1} \in R$; that is, $a = (ab^{-1})b \in \mathfrak{a}$. Thus, $\mathfrak{a} = \mathfrak{a}_i$. In other words, $\mathfrak{a}_i$, $i \in \mathbb{N}_0$, are the only non-zero ideals of R. These ideals form the (infinite) descending chain $\mathfrak{a}_0 = R \supsetneqq \mathfrak{a}_1 = \mathfrak{m} \supsetneqq \mathfrak{a}_2 \supsetneqq \mathfrak{a}_3 \supsetneqq \cdots$.

By definition, $v : K^* \to \mathbb{Z}$ is surjective. Let $x \in R$ be such that $v(x) = 1$. The principal ideal $\langle x \rangle$ is not the unit ideal, satisfies $\mathfrak{a}_1 \supseteq \langle x \rangle \supsetneqq \mathfrak{a}_2$ and hence equals $\mathfrak{a}_1 = \mathfrak{m}$. One can likewise show that $\mathfrak{a}_i = \langle x^i \rangle$ for all $i \in \mathbb{N}$. Thus R is a PID.
[only if] See Exercise 2.133. ■

Recall that the ring $\hat{\mathbb{Z}}_p$ of p-adic integers (Definition 2.111) is a DVR. The field $\mathcal{Q}_p = \mathrm{Q}(\hat{\mathbb{Z}}_p)$ of fractions of $\hat{\mathbb{Z}}_p$ is called the *field of p-adic numbers*. We now explicitly describe a valuation v on $\mathcal{Q}_p$ of which $\hat{\mathbb{Z}}_p$ is the valuation ring. Let the p-adic expansion (Exercises 2.144 and 2.145) of a p-adic integer α be

$$\alpha = k_0 + k_1 p + k_2 p^2 + \cdots . \tag{4.12}$$

A rational integer can be naturally viewed as a p-adic integer with finitely many non-zero terms, that is, one for which $k_i = 0$ except for finitely many $i \in \mathbb{N}_0$. However, a p-adic integer with infinitely many non-zero k_i does not correspond to a rational integer. If in Expansion (4.12) we have $k_1 = k_2 = \cdots = k_{r-1} = 0$, we can write

$$\alpha = p^r (k_r + k_{r+1} p + k_{r+2} p^2 + \cdots) .$$

A p-adic integer is, in general, an infinite series and a representation with *finite precision* looks like

$$k_0 + k_1 p + k_2 p^2 + \cdots + k_s p^s + \mathrm{O}(p^{s+1}) .$$

Arithmetic on p-adic numbers is done like integers written in base p, but from left to right. Thus, for example, if one wants to add two p-adic integers $k_0 + k_1 p + k_2 p^2 + \cdots$ and $k'_0 + k'_1 p + k'_2 p^2 + \cdots$, one may add the base-$p$ integers $\cdots k_2 k_1 k_0$ and $\cdots k'_2 k'_1 k'_0$ in the usual manner till the desired level of precision. A p-adic integer $\alpha = k_0 + k_1 p + k_2 p^2 + \cdots$ is invertible (in $\hat{\mathbb{Z}}_p$) if and only if $k_0 \neq 0$ (Proposition 2.52).

An element $\beta \in \mathcal{Q}_p$ also has a p-adic expansion, but in this case one has to allow terms involving a *finite* number of negative exponents of p. That is to say, we have an expansion of the form

$$\beta = k_{-t}p^{-t} + k_{-t+1}p^{-t+1} + \cdots + k_{-1}p^{-1} + k_0 + k_1 p + k_2 p^2 + \cdots$$

or

$$\beta = p^{-t}(k_{-t} + k_{-t+1}p + \cdots + k_{-1}p^{t-1} + k_0 p^t + k_1 p^{t+1} + k_2 p^{t+2} + \cdots)\,.$$

Of course, if $k_{-t} = k_{-t+1} = \cdots = k_{-1} = 0$, then β is already in $\hat{\mathbb{Z}}_p$.

From the arguments above, it follows that any *non-zero* $\gamma \in \mathcal{Q}_p$ can be written uniquely as $\gamma = p^\delta(\gamma_0 + \gamma_1 p + \gamma_2 p^2 + \cdots)$ with $\delta \in \mathbb{Z}$ and $\gamma_0 \neq 0$. We then set $v(\gamma) := \delta$. It is easy to see that v defines a discrete valuation on $\mathcal{Q}_p$ of which $\hat{\mathbb{Z}}_p$ is the valuation ring. Moreover, since $\gamma_0 + \gamma_1 p + \gamma_2 p^2 + \cdots$ is a unit in $\hat{\mathbb{Z}}_p$, $p = 0 + 1 \cdot p + 0 \cdot p^2 + \cdots$ plays the role of a uniformizer of the DVR $\hat{\mathbb{Z}}_p$. As usual, we write $v(0) = +\infty$.

Now, back to our ECDLP business. Let E be an elliptic curve defined over $\mathbb{F}_p$. Here we consider the case that E is anomalous. We can naturally think of E as a curve over the field $\mathcal{Q}_p$ as well and denote this curve by $\mathcal{E}$. The coordinate-wise application of the canonical surjection $\hat{\mathbb{Z}}_p \to \hat{\mathbb{Z}}_p/p\hat{\mathbb{Z}}_p \cong \mathbb{F}_p$ induces the *reduction homomorphism* $\pi : \mathcal{E}(\mathcal{Q}_p) \to E(\mathbb{F}_p)$. Now, we define the following subgroups of $\mathcal{E}(\mathcal{Q}_p)$:

$$\begin{aligned}
\mathcal{E}_0(\mathcal{Q}_p) &:= \{\mathcal{P} \in \mathcal{E}(\mathcal{Q}_p) \mid \pi(\mathcal{P}) \in E(\mathbb{F}_p)\}\,,\\
\mathcal{E}_1(\mathcal{Q}_p) &:= \{\mathcal{P} \in \mathcal{E}(\mathcal{Q}_p) \mid \pi(\mathcal{P}) = \mathcal{O}\}\,,\\
\mathcal{E}_2(\mathcal{Q}_p) &:= \{\mathcal{P} = (\alpha, \beta) \in \mathcal{E}(\mathcal{Q}_p) \mid v(\alpha) \leqslant -4\} \cup \{\mathcal{O}\}\,.
\end{aligned}$$

It can be shown that $\mathcal{E}_1(\mathcal{Q}_p)$ is a subgroup of $\mathcal{E}_0(\mathcal{Q}_p)$ and $\mathcal{E}_2(\mathcal{Q}_p)$ is a subgroup of $\mathcal{E}_1(\mathcal{Q}_p)$. Furthermore, since E is anomalous, we have

$$\mathcal{E}_0(\mathcal{Q}_p)/\mathcal{E}_1(\mathcal{Q}_p) \cong E(\mathbb{F}_p) \cong (\mathbb{F}_p, +) \cong \mathcal{E}_1(\mathcal{Q}_p)/\mathcal{E}_2(\mathcal{Q}_p)\,.$$

Now, let $P \in E(\mathbb{F}_p)$ and Q a point in the subgroup of $E(\mathbb{F}_p)$ generated by P. Our purpose is to find an integer l such that $Q = lP$. Let $\bar{P}, \bar{Q} \in \mathcal{E}(\mathcal{Q}_p)$ be such that $\pi(\bar{P}) = P$ and $\pi(\bar{Q}) = Q$. It is not difficult to find such points $\bar{P}$ and $\bar{Q}$. For example, if $P = (a, b)$, we can take $\bar{P} = (a, b_0 + b_1 p + b_2 p^2 + \cdots)$, where $b_0 = b$ and $b_1, b_2, \ldots$ are successively obtained by Hensel lifting.

Since $\pi(\bar{Q} - l\bar{P}) = Q - lP = \mathcal{O}$, the point $\bar{R} := \bar{Q} - l\bar{P} \in \mathcal{E}_1(\mathcal{Q}_p)$ and, therefore, $p\bar{R} = p\bar{Q} - pl\bar{P} \in \mathcal{E}_2(\mathcal{Q}_p)$. Now, if we take the so-called *p-adic elliptic logarithm* ψ_p on both sides, we get $\psi_p(p\bar{Q}) - l\psi_p(p\bar{P}) = \psi_p(p\bar{R}) \equiv 0 \pmod{p^2}$, whence it follows that

$$l \equiv \frac{\psi_p(p\bar{Q})}{\psi_p(p\bar{P})} \pmod{p}\,,$$

provided that $\psi_p(p\bar{P})$ is invertible modulo p. The function ψ_p can be easily calculated. Therefore, this gives a very efficient probabilistic algorithm for computing discrete logarithms over anomalous elliptic curves. Here the most time-consuming step is the linear-time computation of the points $p\bar{P}$ and $p\bar{Q}$. For further details on the algorithm (like the computation of $\bar{P}$ and $\bar{Q}$ from P and Q, and the definition of p-adic elliptic logarithms), see Blake et al. [24] and Silverman [275].

** 4.5.3 The Xedni Calculus Method

Joseph Silverman's *xedni calculus method* (XCM) is a recent algorithm for solving the ECDLP in an arbitrary elliptic curve over a finite field. The algorithm is based on some deep mathematical conjectures and heuristic ideas. However, its performance has been experimentally established to be poor. Here we give a sketchy description of the XCM. For simplicity, we concentrate on elliptic curves over prime fields $\mathbb{F}_p$ only.

The basic idea of the XCM is to *lift* an elliptic curve E over $\mathbb{F}_p$ to a curve $\mathcal{E}$ over $\mathbb{Q}$. In view of this, we start with a couple of important results regarding elliptic curves over $\mathbb{Q}$ (or, more generally, over a number field). See Silverman [275], for example, for the proofs.

Let $\mathcal{E}$ be an elliptic curve defined over a number field K.

Theorem 4.3 **Mordell–Weil theorem** The group $\mathcal{E}(K)$ is finitely generated.

The group structure of $\mathcal{E}(K)$ is made explicit by the next theorem. Note that the elements of $\mathcal{E}(K)$ of finite order form a subgroup $\mathcal{E}_{\rm tors}(K)$ of $\mathcal{E}(K)$, called the *torsion subgroup* of $\mathcal{E}(K)$ (Exercise 4.26).

Theorem 4.4 $\mathcal{E}(K) \cong \mathcal{E}_{\rm tors}(K) \times \mathbb{Z}^\rho$ for some $\rho \in \mathbb{N}_0$.

The non-negative integer ρ of Theorem 4.4 is called the *rank* of $\mathcal{E}(K)$.

Now, let E be an elliptic curve defined over a prime field $\mathbb{F}_p$, $P \in E(\mathbb{F}_p)$ and Q a multiple of P. Our task is to compute an integer $l \in \mathbb{Z}$ such that $Q = lP$. We assume that E is defined by a suitable Weierstrass equation. We consider the projective coordinates of points on $E(\mathbb{F}_p)$. Let n denote the cardinality of $E(\mathbb{F}_p)$.

The basic idea of the XCM is to select r points $R_{p,1}, \ldots, R_{p,r} \in E(\mathbb{F}_p)$, compute an elliptic curve $\mathcal{E}$ defined over $\mathbb{Q}$ and points $S_1, \ldots, S_r \in \mathcal{E}(\mathbb{Q})$ such that modulo p the curve $\mathcal{E}$ reduces to E and the points $S_1, \ldots, S_r$ to $R_{p,1}, \ldots, R_{p,r}$. If the rank of $\mathcal{E}$ is small, then the points $S_2, \ldots, S_r$ are expected to be linearly dependent. Computing a non-trivial linear dependency among $S_2, \ldots, S_r$ gives a linear dependency among $R_{p,1}, \cdots, R_{p,r}$, which in turn yields $\mathrm{ind}_P Q$ with high probability. The details are now explained. For r points $L_i := [h_i, k_i, l_i]$, $i = 1, \ldots, r$, we use the notation:

$$B(L_1, \ldots, L_r) := \begin{pmatrix} h_1^3 & h_1^2k_1 & h_1k_1^2 & k_1^3 & h_1^2l_1 & h_1k_1l_1 & k_1^2l_1 & h_1l_1^2 & k_1l_1^2 & l_1^3 \\ h_2^3 & h_2^2k_2 & h_2k_2^2 & k_2^3 & h_2^2l_2 & h_2k_2l_2 & k_2^2l_2 & h_2l_2^2 & k_2l_2^2 & l_2^3 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ h_r^3 & h_r^2k_r & h_rk_r^2 & k_r^3 & h_r^2l_r & h_rk_rl_r & k_r^2l_r & h_rl_r^2 & k_rl_r^2 & l_r^3 \end{pmatrix}.$$

We start by fixing an integer r, $4 \leqslant r \leqslant 9$. We then choose r random pairs (s_i, t_i) of integers and compute the points

$$R_{p,i} := s_iP - t_iQ \in E(\mathbb{F}_p)\,.$$

We now apply a change of coordinates of the form

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} := \begin{pmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} \\ \alpha_{31} & \alpha_{32} & \alpha_{33} \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \qquad \textbf{(4.13)}$$

so that the first four of the points $R_{p,i}$ become $R_{p,1} = [1,0,0]$, $R_{p,2} = [0,1,0]$, $R_{p,3} = [0,0,1]$ and $R_{p,4} = [1,1,1]$. This change of coordinates fails if some three of the four

points $R_{p,1}$, $R_{p,2}$, $R_{p,3}$ and $R_{p,4}$ sum to $\mathcal{O}$. But in that case the desired index $\mathrm{ind}_P\, Q$ can be computed with high probability. If, for example, $R_{p,1} + R_{p,2} + R_{p,3} = \mathcal{O}$, then we have $(s_1 + s_2 + s_3)P = (t_1 + t_2 + t_3)Q$ and, therefore, if $\gcd(t_1 + t_2 + t_3, n) = 1$, then $\mathrm{ind}_P\, Q \equiv (t_1 + t_2 + t_3)^{-1}(s_1 + s_2 + s_3) \pmod{n}$. On the other hand, if $\gcd(t_1 + t_2 + t_3, n) \neq 1$, we repeat with a different set of pairs (s_i, t_i).

Henceforth, we assume that the change of coordinates, as given in Equation (4.13), is successful. This transforms the equation for E to a general cubic equation:

$$\begin{aligned} C_p : u_{p,1}X^3 + u_{p,2}X^2Y + u_{p,3}XY^2 + u_{p,4}Y^3 + u_{p,5}X^2Z + \\ u_{p,6}XYZ + u_{p,7}Y^2Z + u_{p,8}XZ^2 + u_{p,9}YZ^2 + u_{p,10}Z^3 \;=\; 0\,. \end{aligned}$$

Now, we carry out a step that heuristically ensures that the curve $\mathcal{E}$ over $\mathbb{Q}$ (that we are going to construct) has a small rank. We choose a product M of small primes with $p \not| M$, a cubic curve

$$\begin{aligned} C_M : u_{M,1}X^3 + u_{M,2}X^2Y + u_{M,3}XY^2 + u_{M,4}Y^3 + u_{M,5}X^2Z + u_{M,6}XYZ \\ + u_{M,7}Y^2Z + u_{M,8}XZ^2 + u_{M,9}YZ^2 + u_{M,10}Z^3 \equiv 0 \pmod{M} \end{aligned}$$

over $\mathbb{Z}_M$ and points $R_{M,1}, \ldots, R_{M,r}$ on C_M and with coordinates in $\mathbb{Z}_M$. The first four points should be $R_{M,1} = [1, 0, 0]$, $R_{M,2} = [0, 1, 0]$, $R_{M,3} = [0, 0, 1]$ and $R_{M,4} = [1, 1, 1]$. We have to ensure also that for every prime divisor q of M, the matrix $B(R_{M,1}, \ldots, R_{M,r})$ has maximal rank modulo q. In practice, it is easier to choose the points $R_{M,1}, \ldots, R_{M,r}$ first and then compute a curve C_M passing through these points by solving a set of linear equations in the coefficients $u_{M,1}, \ldots, u_{M,10}$ of C_M. The curve C_M should be so chosen that it has the minimum possible number of solutions modulo M. This, in conjunction with some deep conjectures in the theory of elliptic curves, guarantees that the curve $\mathcal{E}$ that we will construct shortly will have a rank less than the expected value.

We now combine the curves C_p and C_M as follows. Using the Chinese remainder theorem, we compute integers $u'_1, \ldots, u'_{10}$ such that $u'_i \equiv u_{p,i} \pmod{p}$ and $u'_i \equiv u_{M,i} \pmod{M}$ for each $i = 1, \ldots, 10$. Similarly, we compute points $R_1, \ldots, R_r$ with integer coefficients such that $R_i \equiv R_{p,i} \pmod{p}$ and $R_i \equiv R_{M,i} \pmod{M}$ for each $i = 1, \ldots, r$, where congruence of points stands for coordinate-wise congruence. Here we have $R_1 = [1, 0, 0]$, $R_2 = [0, 1, 0]$, $R_3 = [0, 0, 1]$ and $R_4 = [1, 1, 1]$.

Clearly, the points $R_1, \ldots, R_r$ are lifts of the points $R_{p,1}, \ldots, R_{p,r}$ respectively, whereas the cubic curve

$$\begin{aligned} \mathcal{C}' : u'_1X^3 + u'_2X^2Y + u'_3XY^2 + u'_4Y^3 + u'_5X^2Z + \\ u'_6XYZ + u'_7Y^2Z + u'_8XZ^2 + u'_9YZ^2 + u'_{10}Z^3 \;=\; 0 \end{aligned}$$

over $\mathbb{Z}_{pM}$ is a lift of E. However, $\mathcal{C}'$, treated as a curve over $\mathbb{Q}$, need not pass through the points $R_1, \ldots, R_r$. In order to ensure this last condition, we modify the coefficients $u'_1, \ldots, u'_{10}$ of $\mathcal{C}'$ to the (small integer) coefficients $u_1, \ldots, u_{10}$ by solving the system of linear equations

$$B(R_1, \ldots, R_r) \begin{pmatrix} u_1 \\ \vdots \\ u_{10} \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}$$

subject to the condition that $u_i \equiv u_i' \pmod{pM}$ for each $i = 1, \ldots, 10$. The resulting cubic curve

$$\mathcal{C} : u_1X^3 + u_2X^2Y + u_3XY^2 + u_4Y^3 + u_5X^2Z + u_6XYZ + u_7Y^2Z + u_8XZ^2 + u_9YZ^2 + u_{10}Z^3 = 0$$

over $\mathbb{Q}$ evidently continues to be a lift of E.

Now, we apply a change of coordinates in order to transfer $\mathcal{C}$ to the standard Weierstrass equation

$$\mathcal{E} : Y^2 + a_1XY + a_3Y = X^3 + a_2X^2 + a_4X + a_6$$

with integer coefficients a_i. This transformation changes the points $R_1, \ldots, R_r$ to the points $S_1, \ldots, S_r$. One should also ensure that $S_1 = \mathcal{O}$.

Finally, we check if $S_2, \ldots, S_r$ are linearly dependent. If so, we determine a (non-trivial) relation $n_2S_2 + \cdots + n_rS_r = \mathcal{O}$ with $n_i \in \mathbb{Z}$. This corresponds to the relation $n_1R_{p,1} + n_2R_{p,2} + \cdots + n_rR_{p,r} = \mathcal{O}$, where $n_1 := -(n_2 + \cdots + n_r)$, that is, $sP = tQ$ with $s := n_1s_1 + \cdots + n_rs_r$ and $t := n_1t_1 + \cdots + n_rt_r$. If $\gcd(t, n) = 1$, we have $\operatorname{ind}_P Q \equiv t^{-1}s \pmod{n}$.

On the other hand, if $S_2, \ldots, S_r$ are linearly independent or if $\gcd(t, n) > 1$, then the lifted data fail to compute $\operatorname{ind}_P Q$. In that case, we repeat the entire process by selecting new pairs (s_i, t_i) and/or new points $R_{M,1}, \ldots, R_{M,r}$.

This completes our description of the XCM. See Silverman [277] for further details. No rigorous or heuristic analysis of the running time of the XCM is available in the literature. Practical experience (reported in Jacobson et al. [139]) shows that the algorithm is rather impractical. The predominant cause for failure of a trial of the XCM is that the probability that the points $S_2, \ldots, S_r$ are linearly dependent is amazingly low. Suitable choices of the curve C_M help us to construct curves $\mathcal{E}$ of low rank, but not low enough, in general, to render $S_2, \ldots, S_r$ linearly dependent. Larger values of r are expected to increase the probability of success in each trial, but it is not clear how to handle the values $r > 9$. Nevertheless, the XCM is a radically new idea to solve the ECDLP. As Joseph Silverman [277] says, "some of the ideas may prove useful in future work on ECDLP".

EXERCISE SET 4.5

4.24 Let K be a field, $m \in \mathbb{N}$ and $\mu_m := \{a \in \bar{K} \mid a^m = 1\}$. Elements of μ_m are called the *m-th roots of unity*. Prove the following assertions.

(a) μ_m is a subgroup of $(\bar{K}^*, \cdot)$.

(b) If $\operatorname{char} K = 0$, then $\#\mu_m = m$. **[H]**

(c) If $p := \operatorname{char} K > 0$, then $\#\mu_m = m/p^{\mathrm{v}_p(m)}$. **[H]**

(d) μ_m is cyclic. **[H]**

(e) The set $\mu := \bigcup_{m \in \mathbb{N}} \mu_m \subseteq \bar{K}^*$ is a subgroup of $\bar{K}^*$.

4.25 We use the notations of the last exercise and assume that $\#\mu_m = m$, that is, either $\operatorname{char} K = 0$ or $p := \operatorname{char} K > 0$ is coprime to m. In this case, a generator of μ_m is called a *primitive m-th root of unity*. If $\omega \in \mu_m$ is a primitive m-th root of unity and

$\omega^r = 1$ for some $r \in \mathbb{N}$, then evidently $m|r$. In particular, m is the smallest of the exponents $r \in \mathbb{N}$ such that $\omega^r = 1$. The (monic) polynomial

$$\Phi_m(X) = \prod_{\omega}(X - \omega) \in \bar{K}[X]\,,$$

where the product runs over all primitive m-th roots of unity, is called the *m-th cyclotomic polynomial* (over K). Clearly, $\deg \Phi_m(X) = \phi(m)$ (where ϕ is Euler's totient function).

(a) Show that $X^m - 1 = \prod_{d|m} \Phi_d(X)$. **[H]** Use the Möbius inversion formula to deduce that $\Phi_m(X) = \prod_{d|m}(X^d - 1)^{\mu(m/d)}$, where μ is the Möbius function. Conclude that $\Phi_m(X) \in K[X]$.

(b) If m is a prime, show that $\Phi_m(X) = X^{m-1} + \cdots + X + 1$.

(c) Let $m \neq 1$ be odd and $\operatorname{char} K \neq 2$. Show that $\Phi_{2m}(X) = \Phi_m(-X)$. **[H]**

(d) Show that if $K = \mathbb{F}_q$, l is the (multiplicative) order of q modulo m and if ω is a primitive m-th root of unity, then $[K(\omega) : K] = l$. **[H]** In particular, Φ_m is a product of $\phi(m)/l$ (distinct) irreducible polynomials each of degree l.

4.26 **(a)** Let G be an (additive) Abelian group (not necessarily finite). Show that the subset

$$G_{\text{tors}} := \{a \in G \mid na = 0 \text{ for some } n \in \mathbb{N}\} \subseteq G$$

is a subgroup of G. G_{tors} is called the *torsion subgroup* of G and the elements of G_{tors} are called *torsion elements* of G. An element $a \in G$ is a torsion element of G if and only if a is of finite order.

(b) Let $\mathcal{E}$ be an elliptic curve defined over a number field K. Show that the torsion subgroup $\mathcal{E}_{\text{tors}}(K)$ of $\mathcal{E}(K)$ is finite. **[H]**

(c) Let $\mathcal{E}$ and K be as in Part (b). Show that $\mathcal{E}_{\text{tors}}(\bar{K})$ is not finite. **[H]**

** 4.6 The Hyperelliptic Curve Discrete Logarithm Problem

The hyperelliptic curve discrete logarithm problem (HECDLP) has attracted less research attention than the ECDLP. Surprisingly, however, there exist subexponential (index calculus) algorithms for solving the HECDLP over curves of large genus. Adleman, DeMarrais and Huang first proposed such an algorithm [2] (which we will refer to as the ADH algorithm). Enge [86] suggested some modifications of the ADH algorithm and provided rigorous analysis of its running time. Gaudry [105] simplified the ADH algorithm and even implemented it. Gaudry's experimentation suggests that it is feasible to compute discrete logarithms in Jacobians of almost cryptographic sizes, given that the genus of the underlying curve is high (say $\geqslant 6$). Enge and Gaudry [87] proved rigorously that as long as the genus g is greater than $\ln q$ ($\mathbb{F}_q$ being the field over which the curve is defined), the ADH algorithm (and its improvements) run in time $L(q^g, 1/2, \sqrt{2})$.

In what follows, we outline Gaudry's version of the ADH algorithm and refer to this as the ADH–Gaudry algorithm. Let $C : Y^2 + u(X)Y = v(X)$ be a hyperelliptic curve of genus g defined over a finite field $\mathbb{F}_q$. We assume that the cardinality of the Jacobian

$\mathbb{J}(C) = \mathbb{J}_{\mathbb{F}_q}(C)$ is known and has a suitably large prime divisor m. We assume further that a reduced divisor $\alpha \in \mathbb{J}(C)$ of order m is available, and we want to compute the discrete logarithm $\mathrm{ind}_\alpha \beta$ of $\beta \in \langle\alpha\rangle$ with respect to α.

4.6.1 Choosing the Factor Base

Recall that every reduced divisor $D \in \mathbb{J}(C)$ can be written uniquely as $D = P_1 + \cdots + P_l - l\mathcal{O}$, $l \leqslant g$, where for $i \neq j$ the points P_i and P_j are not opposite of each other. Only ordinary points (not special points) may appear more than once in the list $P_1, \ldots, P_l$. We also know that such a divisor can be represented by a pair of unique polynomials $a, b \in \mathbb{F}_q[X]$ satisfying $\deg b < \deg a \leqslant g$ and $a|(b^2 + bu - v)$. In that case, we write $D = \mathrm{Div}(a, b)$. What interests us is the fact that the roots of the polynomial a are precisely the X-coordinates of the points $P_1, \ldots, P_l$. This fact leads to the very useful concepts of prime divisors and smooth divisors.

Definition 4.4 A divisor $D = \mathrm{Div}(a, b) \in \mathbb{J}(C) = \mathbb{J}_{\mathbb{F}_q}(C)$ is called *prime*, if the polynomial $a(X) \in \mathbb{F}_q[X]$ is irreducible (that is, prime) over $\mathbb{F}_q$.

For an arbitrary divisor $D = \mathrm{Div}(a, b) \in \mathbb{J}(C)$, let $a = a_1 \cdots a_r$ be the factorization of a into irreducible polynomials a_i over $\mathbb{F}_q$. There exist polynomials $b_1, \ldots, b_r \in \mathbb{F}_q[X]$ such that $D = \sum_{i=1}^{r} D_i$, where $D_i := \mathrm{Div}(a_i, b_i)$. In that case, the (prime) divisors $D_1, \ldots, D_r$ are called the *prime divisors of* D. Moreover, if $\deg a_i \leqslant \delta$ for all $i = 1, \ldots, r$ and for some $\delta \in \mathbb{N}$, then D is called δ-smooth. In particular, $D = \mathrm{Div}(a, b)$ is 1-smooth if and only if a splits completely over $\mathbb{F}_q$. □

In order to set up a factor base B, we predetermine a smoothness bound δ and let B consist of all the prime divisors $\mathrm{Div}(a, b) \in \mathbb{J}(C)$ with $\deg a \leqslant \delta$. For simplicity, we take $\delta = 1$. This is indeed a practical choice, when the genus g is not too large (say, $g \leqslant 9$). Let $a(X) = X - h \in \mathbb{F}_q[X]$ be an (irreducible) polynomial of degree 1. In order to find out $b \in \mathbb{F}_q[X]$ such that $\mathrm{Div}(a, b)$ is a prime divisor, we first see that $\deg b < \deg a$, that is, $b \in \mathbb{F}_q$. Furthermore, $a|(b^2 + bu - v)$: that is, $b^2 + bu - v \equiv 0 \pmod{X - h}$; that is, $b^2 + bu(h) - v(h) = 0$. Thus, the desired values of $b \in \mathbb{F}_q$, if existent, can be found by solving a quadratic equation over $\mathbb{F}_q$. There are q irreducible polynomials $a \in \mathbb{F}_q[X]$ of degree 1 and for each such a there are either two or no solutions for $b \in \mathbb{F}_q$. Assuming that both these possibilities are equally likely, we conclude that the size of the factor base is $\approx q$.

4.6.2 Checking the Smoothness of a Divisor

In order to check for the smoothness of a divisor $D = \mathrm{Div}(a, b) \in \mathbb{J}(C)$ over the factor base B, we first factor a over $\mathbb{F}_q$. Under the assumption that $\delta = 1$, the divisor D is smooth if and only if a splits completely over $\mathbb{F}_q$. Let us write $a(X) = (X - h_1) \cdots (X - h_l)$, $h_i \in \mathbb{F}_q$. Then for some $k_1, \ldots, k_l \in \mathbb{F}_q$ we have $D = \sum_{i=1}^{l} D_i$, where $D_i := \mathrm{Div}(X - h_i, k_i)$. We may use trial divisions (that is, trial subtractions in this additive setting) by elements of B in order to determine the prime divisors $D_1, \ldots, D_l$ of D. Proposition 4.5 establishes the probability that a randomly chosen element of $\mathbb{J}(C)$ is smooth.

Proposition 4.5 For $q \gg 4g^2$, there are approximately $q^g/g!$ (1-)smooth divisors in $\mathbb{J}(C)$. In particular, the probability that a randomly chosen divisor in $\mathbb{J}(C)$ is smooth is approximately $1/g!$.

The assumption $q \gg 4g^2$ is practical, since we usually employ curves of (fixed) small genus g over finite fields $\mathbb{F}_q$ of medium sizes. For example, Koblitz [154] proposed the curve $Y^2 + Y = X^{13}$ of genus $g = 6$ over the prime field $\mathbb{F}_{5026243}$. An interesting consequence of the last proposition is that the proportion of smooth divisors in $\mathbb{J}(C)$ depends only on the genus g of C (and not on q).

4.6.3 The Algorithm

Now, we have all the machinery required to describe the basic version of the index calculus method for computing $\mathrm{ind}_\alpha \beta$ in $\mathbb{J}(C)$. In the first stage, we choose a random $j \in \{1, \ldots, m-1\}$ and compute the (reduced) divisor $j\alpha$ and check if $j\alpha$ is smooth over the factor base B. Every smooth $j\alpha$ gives a relation: that is, a linear congruence modulo m involving the (unknown) indices of the elements of B to the base α. After sufficiently many (say, $\geqslant 2(\#B)$) such relations are found, the system of linear congruences collected is expected to be of full rank and is solved modulo m. This gives us the indices of the elements of the factor base. Each congruence collected above contains at most g non-zero coefficients and so the system is necessarily sparse. In the second stage, we find out a single random j for which $\beta + j\alpha$ is smooth. The database prepared in the first stage then immediately gives $\mathrm{ind}_\alpha \beta$.

The Hasse–Weil Bounds (3.8) on p 226 show that the cardinality of $\mathbb{J}(C)$ is approximately q^g. Thus $\mathrm{O}(g \log q)$ bits are needed to represent an element of $\mathbb{J}(C)$. This fact is consistent with the representation of reduced divisors by pairs of polynomials. Gaudry [105] calculates that this variant of the ICM does $\mathrm{O}(q^2 + g!q)$ operations, each of which takes polynomial time in the input size $g \log q$. If g is considered to be constant, the running time becomes $\mathrm{O}(q^2 \log^t q)$ (that is, $\tilde{\mathrm{O}}(q^2)$) for some real $t > 0$. A square root method on $\mathbb{J}(C)$ runs in (expected) time $\tilde{\mathrm{O}}(q^{g/2})$. Thus for $g > 4$ the index calculus method performs better than the square root methods. Indeed Gaudry's implementation of this algorithm is capable of computing in a few days discrete logs in the curve of genus 6 mentioned above. The Jacobian of this curve is of cardinality $\approx 10^{40}$.

For cryptographic purposes, we should have $\#\mathbb{J}(C) \approx q^g \geqslant 2^{160}$. If we want to take q small (so that multi-precision arithmetic can be avoided), we should choose large values of g. But this choice makes the ADH–Gaudry algorithm quite efficient. For achieving the desired level of security in cryptographic applications, hyperelliptic curves of genus 2, 3 and 4 only are recommended.

4.7 Solving Large Sparse Linear Systems over Finite Rings

So far we have seen many algorithms which require solving large systems of linear equations (or congruences). The number n of unknowns in such systems can be as large as several millions. Standard Gaussian elimination on such a system takes time $\mathrm{O}(n^3)$ and space $\mathrm{O}(n^2)$. There are asymptotically faster algorithms like Strassen's method [292] that takes time $\mathrm{O}(n^{2.807})$ and Coppersmith and Winograd's method [60]

having a running time of $\mathrm{O}(n^{2.376})$. Unfortunately, these asymptotic estimates do not show up in the range of practical interest. Moreover, the space requirements of these asymptotically faster methods are prohibitively high (though still $\mathrm{O}(n^2)$).

Luckily enough, cryptanalytic algorithms usually deal with coefficient matrices that are *sparse*: that is, that have only a small number of non-zero entries in each row. For example, consider the system of linear congruences available from the relation collection stage of an ICM for solving the DLP over a finite field $\mathbb{F}_q$. The factor base consists of a subexponential (in $\lg q$) number of elements, whereas each relation involves at most $\mathrm{O}(\lg q)$ non-zero coefficients. Furthermore, the sparsity of the resulting matrix A is somewhat *structured* in the sense that the columns of A corresponding to larger primes in the factor base tend to have fewer numbers of non-zero entries. In this regard, we refer to the interesting analysis by Odlyzko [225] in connection with the Coppersmith method (Section 4.4.4). Odlyzko took $m = 2n$ equations in n unknown indices and showed that about $n/4$ columns of A are expected to contain only zero coefficients, implying that these variables never occurred in any relation collected. Moreover, about $0.346n$ columns of A are expected to have only single non-zero coefficients.

The sparsity (as well as the structure of the sparsity) of the coefficient matrix A can be effectively exploited and the system can be solved in time $\tilde{\mathrm{O}}(n^2)$. In this section, we describe some special algorithms for large sparse linear systems. In what follows, we assume that we want to compute the unknown n-dimensional column vector $\mathbf{x}$ from the given system of equations

$$A\mathbf{x} = \mathbf{b}\,,$$

where A is an $m \times n$ matrix, $m \geqslant n$, and where $\mathbf{b}$ is a non-zero m-dimensional column vector. Though this is not the case in general, we will often assume for the sake of simplicity that A has full rank (that is, n). We write vectors as column vectors, that is, an l-dimensional vector $\mathbf{v}$ with elements $v_1, \ldots, v_l$ is written as $\mathbf{v} = (v_1\ v_2\ \ldots\ v_l)^{\mathrm{t}}$, where the superscript $^{\mathrm{t}}$ denotes matrix transpose.

Before we proceed further, some comments are in order. First note that our system of equations is often one over the finite ring $\mathbb{Z}_r$ which is not necessarily a field. Most of the methods we describe below assume that $\mathbb{Z}_r$ is a field, that is, r is a prime. If r is composite, we can do the following. First, assume that the prime factorization $r = p_1^{\alpha_1} \cdots p_s^{\alpha_s}$, $\alpha_i > 0$, of r is known. In that case, we first solve the system over the fields $\mathbb{Z}_{p_i}$ for $i = 1, \ldots, s$. Then for each i we *lift* the solution modulo p_i to the solution modulo $p_i^{\alpha_i}$. Finally, all these lifted solutions are combined using the CRT to get the solution modulo r.

Hensel lifting can be used to lift a solution of the system $A\mathbf{x} \equiv \mathbf{b} \pmod{p}$ to a solution of $A\mathbf{x} \equiv \mathbf{b} \pmod{p^\alpha}$, where p is a prime and $\alpha \in \mathbb{N}$. We proceed by induction on α. Let us denote the (or a) solution of $A\mathbf{x} \equiv \mathbf{b} \pmod{p}$ by $\mathbf{x}_1$, which can be computed by solving a system in the field $\mathbb{Z}_p = \mathbb{F}_p$. Now, assume that for some $i \in \{1, \ldots, \alpha - 1\}$ we know (integer) vectors $\mathbf{x}_1, \ldots, \mathbf{x}_i$ such that

$$A(\mathbf{x}_1 + p\mathbf{x}_2 + \cdots + p^{i-1}\mathbf{x}_i) \equiv \mathbf{b} \pmod{p^i}\,. \tag{4.14}$$

We then attempt to compute a vector $\mathbf{x}_{i+1}$ such that

$$A(\mathbf{x}_1 + p\mathbf{x}_2 + \cdots + p^{i-1}\mathbf{x}_i + p^i\mathbf{x}_{i+1}) \equiv \mathbf{b} \pmod{p^{i+1}}\,. \tag{4.15}$$

Congruence (4.14) shows that the elements of $A, \mathbf{x}_1, \ldots, \mathbf{x}_i, \mathbf{b}$ can be so chosen (as integers) that for some vector $\mathbf{y}_i$ we have the equality

$$A(\mathbf{x}_1 + p\mathbf{x}_2 + \cdots + p^{i-1}\mathbf{x}_i) = \mathbf{b} - p^i\mathbf{y}_i$$

in $\mathbb{Z}$. Substituting this in Congruence (4.15) gives $A\mathbf{x}_{i+1} \equiv \mathbf{y}_i \pmod{p}$. Thus the (incremental) vector $\mathbf{x}_{i+1}$ can be obtained by solving a linear system in $\mathbb{Z}_p$.

It, therefore, suffices to know how to solve linear congruences modulo a prime p. However, problems arise, when we do not know the factorization of r (while solving $A\mathbf{x} \equiv \mathbf{b} \pmod{r}$). If r is large, it would be a heavy investment to make attempts to factor r. What can be done instead is the following. First, we use trial divisions to extract the small prime factors of r. We may, therefore, assume that r has no small prime factors. We proceed to solve $A\mathbf{x} \equiv \mathbf{b} \pmod{r}$ *assuming* that r is a prime (that is, that $\mathbb{Z}_r$ is a field). In a field, every non-zero element is invertible. But if r is composite, there are non-zero elements $a \in \mathbb{Z}_r$ which are not invertible (that is, for which $\gcd(a, r) > 1$). If, during the course of the computation, we never happen to meet (and try to invert) such non-zero non-invertible elements, then the computation terminates without any trouble. Otherwise, such an element a corresponds to a non-trivial factor $\gcd(a, r)$ of r. In that case, we have a partial factorization of r and restart solving the system modulo each suitable factor of r.

Some of the algorithms we discuss below assume that A is a symmetric matrix. In our case, this is usually not the case. Indeed we have matrices A which are not even square. Both these problems can be overcome by trying to solve the modified system $A^{\mathrm{t}}A\mathbf{x} = A^{\mathrm{t}}\mathbf{b}$. If A has full rank, this leads to an equivalent system.

If $r = 2$ (as in the case of the QSM for factoring integers), using the special methods is often not recommended. In this case, the elements of A are bits and can be packed compactly in machine words, and addition of rows can be done word-wise (say, 32 bits at a time). This leads to an efficient implementation of ordinary Gaussian elimination, which usually runs faster than the more complicated special algorithms described below, at least for the sizes of practical systems.

In what follows, we discuss some well-known methods for solving large sparse linear systems over finite fields (typically prime fields). In order to simplify notations, we will refrain from writing the matrix equalities as congruences, but treat them as *equations* over the underlying finite fields.

4.7.1 Structured Gaussian Elimination

Structured Gaussian elimination is applied to a sparse system before one of the next three methods is employed to solve the system. If the sparsity of A has some *structures* (as discussed earlier), then structured Gaussian elimination tends to reduce the size of the system considerably, while maintaining the sparsity of the system. We now describe the essential steps of structured Gaussian elimination. Let us define the *weight* of a row or column of a matrix to be the number of non-zero entries in that row or column.

First we delete all the columns (together with the corresponding variables) that have weight 0. These variables never occur in the system and need not be considered at all.

Next we delete all the columns that have weight 1 and the rows corresponding to the non-zero entries in these columns. Each such deleted column correspond to a variable

x_i that appears in exactly one equation. After the rest of the system is solved, the value of x_i is obtained by back substitution. Deleting some rows in the matrix in this step may expose some new columns of weight 1. So this step should be repeated, until all the columns have weight > 1.

Now, choose each row with weight 1. This gives a direct solution for the variable x_i corresponding to the non-zero entry of the row. We then substitute this value of x_i in all the equations where it occurs and subsequently delete the ith column. We repeat this step, until all rows are of weight > 1.

At this point, the system usually has many more equations than variables. We may make the system a square one by throwing away some rows. Since subtracting multiples of rows of higher weights tends to increase the number of non-zero elements in the matrix, we should throw away the rows with higher weights. While discarding the excess rows, we should be careful to ensure that we are not left with a matrix having columns of weight 0. Some columns in the reduced system may again happen to have weight 1. Thus, we have to repeat the above steps again. And again and again and ..., until we are left with a square matrix each row and column of which has weight $\geqslant 2$.

This procedure leads to a system which is usually much smaller than the original system. In a typical example quoted in Odlyzko [225], structured Gaussian elimination reduces a system with $16{,}500$ unknowns to one with less that $1{,}000$ unknowns. The resulting reduced system may be solved using ordinary Gaussian elimination which, for smaller systems, appears to be much faster than the following sophisticated methods.

4.7.2 The Conjugate Gradient Method

The conjugate gradient method was originally proposed to solve a linear system $A\mathbf{x} = \mathbf{b}$ over $\mathbb{R}$ for an $n \times n$ (that is, square) symmetric positive definite matrix A and for a non-zero vector $\mathbf{b}$ and is based on the idea of minimizing the quadratic function $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^{\mathrm{t}}A\mathbf{x} - \mathbf{b}^{\mathrm{t}}\mathbf{x}$. The minimum is attained, when the gradient $\nabla f = A\mathbf{x} - \mathbf{b}$ equals zero, which corresponds to the solution of the given system.

The conjugate gradient method is an iterative procedure. The iterations start with an initial *minimizer* $\mathbf{x}_0$ which can be any n-dimensional vector. As the iterations proceed, we obtain gradually improved minimizers $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots$, until we reach the solution. We also maintain and update two other sequences of vectors $\mathbf{e}_i$ and $\mathbf{d}_i$. The vector $\mathbf{e}_i$ stands for the error $\mathbf{b} - A\mathbf{x}_i$, whereas the vectors $\mathbf{d}_0, \mathbf{d}_1, \ldots$ constitute a set of mutually conjugate (that is, orthogonal) directions. We initialize $\mathbf{e}_0 = \mathbf{d}_0 = \mathbf{b} - A\mathbf{x}_0$ and for $i = 0, 1, \ldots$ repeat the steps of Algorithm 4.8, until $\mathbf{e}_i = 0$. We denote the inner product of two vectors $\mathbf{v} = (v_1\ v_2\ \ldots\ v_n)^{\mathrm{t}}$ and $\mathbf{w} = (w_1\ w_2\ \ldots\ w_n)^{\mathrm{t}}$ by $\langle \mathbf{v}, \mathbf{w} \rangle := \sum_{i=1}^{n} v_i w_i$.

Algorithm 4.8 **An iteration in the conjugate gradient method**

$a_i := \langle \mathbf{e}_i, \mathbf{e}_i \rangle / \langle \mathbf{d}_i, A\mathbf{d}_i \rangle$.
$\mathbf{x}_{i+1} := \mathbf{x}_i + a_i \mathbf{d}_i$.
$\mathbf{e}_{i+1} := \mathbf{e}_i - a_i A\mathbf{d}_i$.
$b_i := \langle \mathbf{e}_{i+1}, \mathbf{e}_{i+1} \rangle / \langle \mathbf{e}_i, \mathbf{e}_i \rangle$.
$\mathbf{d}_{i+1} := \mathbf{e}_{i+1} + b_i \mathbf{d}_i$.

This method computes a set of mutually orthogonal directions $\mathbf{d}_0, \mathbf{d}_1, \ldots$, and hence it has to stop after at most $n-1$ iterations, since we run out of new orthogonal directions after $n-1$ iterations. Provided that we work with infinite precision, we must eventually obtain $\mathbf{e}_i = \mathbf{0}$ for some i, $0 \leqslant i \leqslant n-1$.

If A is sparse, that is, if each row of A has $\mathrm{O}(\log^c n)$ non-zero entries, c being a positive constant, then the product $A\mathbf{d}_i$ can be computed using $\tilde{\mathrm{O}}(n)$ field operations. Other operations clearly meet this bound. Since at most $n-1$ iterations are necessary, the conjugate gradient method terminates after performing $\tilde{\mathrm{O}}(n^2)$ field operations.

We face some potential problems, when we want to apply this method to solve a system over a finite field $\mathbb{F}_q$. First, the matrix A is usually not symmetric and need not even be square. This problem can be avoided by solving the system $A^{\mathrm{t}}A\mathbf{x} = A^{\mathrm{t}}\mathbf{b}$. The new coefficient matrix $A^{\mathrm{t}}A$ may be non-sparse (that is, dense). So instead of computing and working with $A^{\mathrm{t}}A$ explicitly, we compute the product $(A^{\mathrm{t}}A)\mathbf{d}_i$ as $A^{\mathrm{t}}(A\mathbf{d}_i)$, that is, we avoid multiplication by a (possibly) dense matrix at the cost of multiplications by two sparse matrices.

The second difficulty with a finite field $\mathbb{F}_q$ is that the question of minimizing an $\mathbb{F}_q$-valued function makes hardly any sense (and so does positive definiteness of a matrix over $\mathbb{F}_q$). However, the conjugate gradient method is essentially based on the generation of a set of mutually orthogonal vectors $\mathbf{d}_0, \mathbf{d}_1, \ldots$. This concept continues to make sense in the setting of a finite field.

If A is a *real* positive definite matrix, we cannot have $\langle \mathbf{d}_i, A\mathbf{d}_i \rangle = 0$ for a non-zero vector $\mathbf{d}_i$. But this condition need not hold for a matrix A over $\mathbb{F}_q$. Similarly, we may have a non-zero error vector $\mathbf{e}_i$ over $\mathbb{F}_q$, for which $\langle \mathbf{e}_i, \mathbf{e}_i \rangle = 0$. (Again this is not possible for real vectors.) So for the iterations over $\mathbb{F}_q$ (more precisely, the computations of a_i and b_i) to proceed gracefully, all that we can hope for is that before reaching the solution we never hit a non-zero direction vector $\mathbf{d}_i$ for which $\langle \mathbf{d}_i, A\mathbf{d}_i \rangle = 0$ nor a non-zero error vector $\mathbf{e}_i$ for which $\langle \mathbf{e}_i, \mathbf{e}_i \rangle = 0$. If q is sufficiently large and if the initial minimizer $\mathbf{x}_0$ is sufficiently randomly chosen, then the probability of encountering such a bad $\mathbf{d}_i$ or $\mathbf{e}_i$ is rather low and as a result the method is very likely to terminate without problems. If, by a terrible stroke of bad luck, we have to abort the computation prematurely, we should restart the procedure with a new random initial vector $\mathbf{x}_0$. If q is small (say $q = 2$ as in the case of the QSM), it is a neater idea to select the entries of the initial vector $\mathbf{x}_0$ from a field extension $\mathbb{F}_{q^s}$ and work in this extension. The eventual solution we will reach at will be in $(\mathbb{F}_q)^n$, but working in the larger field decreases the possibility of an attempt of division by 0.

There is, however, a brighter side of using a finite field $\mathbb{F}_q$ in place of $\mathbb{R}$, namely every calculation we perform in $\mathbb{F}_q$ is *exact*, and we do not have to bother about a criterion for determining whether an error vector $\mathbf{e}_i$ is zero or about the conditioning of the matrix A. One of the biggest headaches of numerical analysis is absent here.

4.7.3 The Lanczos Method

The Lanczos method is another iterative method quite similar to the conjugate gradient method. The basic difference between these methods lies in the way by which the mutually conjugate directions $\mathbf{d}_0, \mathbf{d}_1, \ldots$ are generated. For the Lanczos method, we start with the initializations: $\mathbf{d}_0 := \mathbf{b}$, $\mathbf{v}_1 := A\mathbf{d}_0$, $\mathbf{d}_1 := \mathbf{v}_1 - \mathbf{d}_0(\mathbf{v}_1^{\mathrm{t}}A\mathbf{d}_0)/(\mathbf{d}_0^{\mathrm{t}}A\mathbf{d}_0)$,

$a_0 := (\mathbf{d}_0^{\mathrm{t}}\mathbf{d}_0)/(\mathbf{d}_0^{\mathrm{t}}A\mathbf{d}_0)$, $\mathbf{x}_0 = a_0\mathbf{d}_0$. Then, for $i = 1, 2, \ldots$, we repeat the steps in Algorithm 4.9 as long as $\mathbf{d}_i^{\mathrm{t}}A\mathbf{d}_i \neq \mathbf{0}$.

Algorithm 4.9 **An iteration in the Lanczos method**

$\mathbf{v}_{i+1} := A\mathbf{d}_i$.
$\mathbf{d}_{i+1} := \mathbf{v}_{i+1} - \mathbf{d}_i(\mathbf{v}_{i+1}^{\mathrm{t}}A\mathbf{d}_i)/(\mathbf{d}_i^{\mathrm{t}}A\mathbf{d}_i) - \mathbf{d}_{i-1}(\mathbf{v}_{i+1}^{\mathrm{t}}A\mathbf{d}_{i-1})/(\mathbf{d}_{i-1}^{\mathrm{t}}A\mathbf{d}_{i-1})$.
$a_i := (\mathbf{d}_i^{\mathrm{t}}\mathbf{b})/(\mathbf{d}_i^{\mathrm{t}}A\mathbf{d}_i)$.
$\mathbf{x}_i := \mathbf{x}_{i-1} + a_i\mathbf{d}_i$.

If A is a real positive definite matrix, the termination criterion $\mathbf{d}_i^{\mathrm{t}}A\mathbf{d}_i = \mathbf{0}$ is equivalent to the condition $\mathbf{d}_i = \mathbf{0}$. When this is satisfied, the vector $\mathbf{x}_{i-1}$ equals the desired solution $\mathbf{x}$ of the system $A\mathbf{x} = \mathbf{b}$. Since $\mathbf{d}_0, \mathbf{d}_1, \ldots$ are mutually orthogonal, the process must stop after at most $n - 1$ iterations. Therefore, for a sparse matrix A, the entire procedure performs $\tilde{\mathrm{O}}(n^2)$ field operations.

The problems we face with the Lanczos method applied to a system over $\mathbb{F}_q$ are essentially the same as those discussed in connection with the conjugate gradient method. The problem with a non-symmetric and/or non-square matrix A is solved by multiplying the system by A^{t}. Instead of working with $A^{\mathrm{t}}A$ explicitly, we prefer to multiply separately by A and A^{t}.

The more serious problem with a system over $\mathbb{F}_q$ is that of encountering a non-zero direction vector $\mathbf{d}_i$ with $\mathbf{d}_i^{\mathrm{t}}A\mathbf{d}_i = \mathbf{0}$. If it happens, we have to abort the computation prematurely. In order to restart the procedure, we try to solve the system $BA\mathbf{x} = B\mathbf{b}$, where B is a *diagonal* matrix whose diagonal elements are chosen randomly from the non-zero elements of the field $\mathbb{F}_q$ or of some suitable extension $\mathbb{F}_{q^s}$ (if q is small).

4.7.4 The Wiedemann Method

The Wiedemann method for solving a sparse system $A\mathbf{x} = \mathbf{b}$ over $\mathbb{F}_q$ uses ideas different from those employed by the other methods discussed so far. For the sake of simplicity, we assume that A is a square non-singular matrix (not necessarily symmetric). The Wiedemann method tries to compute the minimal polynomial $\mu_A(X) := \sum_{i=0}^{d} c_i X^i \in \mathbb{F}_q[X]$, $d \leqslant n$, of A. To that end, one selects a small positive integer l in the range $10 \leqslant l \leqslant 20$. For $i \in \mathbb{N}_0$, let $\mathbf{v}_i$ denote the column vector of length l consisting of the first l entries of the vector $A^i\mathbf{b}$. For the working of the Wiedemann method, we need to compute only the vectors $\mathbf{v}_0, \ldots, \mathbf{v}_{2n}$. If A is a sparse matrix, this computation involves a total of $\tilde{\mathrm{O}}(n^2)$ operations in $\mathbb{F}_q$.

Since $\mu_A(A) = 0$, we have $\sum_{i=0}^{d} c_i A^{i+j}\mathbf{b} = \mathbf{0}$ for every $j \in \mathbb{N}_0$. Therefore, for each $k = 1, \ldots, l$ the sequence $v_{0,k}, v_{1,k}, \ldots$ of the k-th entries of $\mathbf{v}_0, \mathbf{v}_1, \ldots$ satisfies the linear recurrence

$$\sum_{i=0}^{d} c_i v_{i+j,k} = 0 \text{ for each } j \in \mathbb{N}_0 \,.$$

But then the minimal polynomial $\mu_k(X)$ of the k-th such sequence is a factor of $\mu_A(X)$. There are methods that compute each $\mu_k(X)$ using $\mathrm{O}(n^2)$ field operations. We then expect to obtain $\mu_A(X) = \mathrm{lcm}(\mu_k(X) \mid 1 \leqslant k \leqslant l)$.

The assumption that A is non-singular is equivalent to the condition that $c_0 \neq 0$. In that case, the solution vector $\mathbf{x} = A^{-1}\mathbf{b} = -c_0^{-1} \sum_{i=1}^{d} c_i A^{i-1}\mathbf{b}$ can be computed using $\tilde{O}(n^2)$ arithmetic operations in the field $\mathbb{F}_q$.

If A is singular, we may find out linear dependencies among the rows of A and subsequently throw away suitable rows. Doing this repeatedly eventually gives us a non-singular A. For further details on the Wiedemann method, see [303].

4.8 The Subset Sum Problem

In this section, we assume that $A := \{a_1, \ldots, a_n\} \subseteq \mathbb{N}$ be a knapsack set. For $s \in \mathbb{N}$, we are required to find out $\epsilon_1, \ldots, \epsilon_n \in \{0, 1\}$ such that $s = \sum_{i=1}^{n} \epsilon_i a_i$, provided that a solution exists. In general, finding such a solution for $\epsilon_1, \ldots, \epsilon_n$ is a very difficult problem.[6] However, if the weights satisfy some specific bounds, there exist polynomial-time algorithms for solving the SSP.

Let us first define an important quantity associated with a knapsack set:

Definition 4.5 The *density* of the knapsack set $A = \{a_1, \ldots, a_n\} \subseteq \mathbb{N}$ is defined to be the real number $d(A) := \frac{n}{\lg\left(\max_{1 \leqslant i \leqslant n} a_i\right)}$. □

If $d(A) > 1$, then there are, in general, more than one solutions for the SSP (provided that there exists one solution). This makes the corresponding knapsack set A unsuitable for cryptographic purposes. So we consider low densities: that is, the case that $d(A) \leqslant 1$.

There are certain algorithms that reduce in polynomial time the problem of finding a solution of the SSP to that of finding a shortest (non-zero) vector in a lattice. Assuming that such a vector is computable in polynomial time, Lagarias and Odlyzko's reduction algorithm [157] solves the SSP in polynomial time with high probability, if $d(A) \leqslant 0.6463$. An improved version of the algorithm adapts to densities $d(A) \leqslant 0.9408$ (see Coster et al. [64] and Coster et al. [65]). The reduction algorithm is easy and will be described in Section 4.8.1. However, it is not known how to efficiently compute a shortest non-zero vector in a lattice. The Lenstra–Lenstra–Lovasz (L^3) polynomial-time lattice basis reduction algorithm [166] provably finds out a non-zero vector whose length is at most the length of a shortest non-zero vector, multiplied by a power of 2. In practice, however, the L^3 algorithm tends to compute a shortest vector quite often. Section 4.8.2 deals with the L^3 lattice basis reduction algorithm.

Before providing a treatment on lattices, let us introduce a particular case of the SSP, which is easily (and uniquely) solvable.

Definition 4.6 A knapsack set $\{a_1, \ldots, a_n\}$ with $a_1 < \cdots < a_n$ is said to be *superincreasing*, if $a_i > \sum_{j=1}^{i-1} a_j$ for all $j = 2, \ldots, n$. □

Algorithm 4.10 solves the SSP for a superincreasing knapsack set in deterministic polynomial time. The proof for the correctness of this algorithm is easy and left to the reader.

[6] In the language of complexity theory, the decision problem of determining whether a solution of the SSP exists is NP-complete.

Algorithm 4.10 **Solving the superincreasing knapsack problem**

Input: A superincreasing knapsack set $\{a_1, \ldots, a_n\}$ with $a_1 < \cdots < a_n$ and $s \in \mathbb{N}$.
Output: The (unique) solution for $\epsilon_1, \ldots, \epsilon_n \in \{0, 1\}$ of $s = \sum_{i=1}^{n} \epsilon_i a_i$, if it exists, failure, otherwise.
Steps:
for $i = n, n-1, \ldots, 1$ {
  if $(s \geqslant a_i)$ { $\epsilon_i := 1$, $s := s - a_i$. } else { $\epsilon_i := 0$. }
}
if $(s = 0)$ { Return $(\epsilon_1, \ldots, \epsilon_n)$. } else { Return "failure". }

4.8.1 The Low-Density Subset Sum Problem

We start by defining a lattice.

Definition 4.7 Let $n, d \in \mathbb{N}$, $d \leqslant n$, and let $\mathbf{v}_1, \ldots, \mathbf{v}_d \in \mathbb{R}^n$ be d linearly independent (non-zero) vectors (that is, n-tuples). The *lattice* L of *dimension* d spanned by $\mathbf{v}_1, \ldots, \mathbf{v}_d$ is the set of all $\mathbb{Z}$-linear combinations of $\mathbf{v}_1, \ldots, \mathbf{v}_d$, that is,

$$L := \{n_1\mathbf{v}_1 + \cdots + n_d\mathbf{v}_d \mid n_1, \ldots, n_d \in \mathbb{Z}\} \subseteq \mathbb{R}^n.$$

We say that $\mathbf{v}_1, \ldots, \mathbf{v}_d$ constitute a *basis* of L. □

In general, a lattice may have more than one basis. We are interested in bases consisting of *short* vectors, where the concept of *shortness* is with respect to the following definition.

Definition 4.8 Let $\mathbf{v} := (v_1, \ldots, v_n)^{\mathrm{t}}$ and $\mathbf{w} := (w_1, \ldots, w_n)^{\mathrm{t}}$ be two n-dimensional vectors in $\mathbb{R}^n$. The *inner product* of $\mathbf{v}$ and $\mathbf{w}$ is defined to be the non-negative real number

$$\langle \mathbf{v}, \mathbf{w} \rangle := v_1 w_1 + \cdots + v_n w_n \,,$$

and the *length* of $\mathbf{v}$ is defined as

$$||\mathbf{v}|| := \sqrt{\langle \mathbf{v}, \mathbf{v} \rangle} = \sqrt{v_1^2 + \cdots + v_n^2} \,. \qquad \square$$

For the time being, let us assume the availability of a *lattice oracle* which, given a lattice, returns a shortest non-zero vector in the lattice. The possibilities for realizing such an oracle will be discussed in the next section.

Consider the subset sum problem with the knapsack set $A := \{a_1, \ldots, a_n\}$ and let B be an upper bound on the weights (that is, each $a_i \leqslant B$). For $s \in \mathbb{N}$, we are supposed to find out $\epsilon_1, \ldots, \epsilon_n \in \{0, 1\}$ such that $s = \sum_{i=1}^{n} \epsilon_i a_i$. Let L be the $n+1$-dimensional lattice in $\mathbb{R}^{n+1}$ generated by the vectors

$$\begin{aligned}
\mathbf{v}_1 &= (1, 0, \ldots, 0, Na_1)^{\mathrm{t}}, \\
\mathbf{v}_2 &= (0, 1, \ldots, 0, Na_2)^{\mathrm{t}}, \\
&\vdots \\
\mathbf{v}_n &= (0, 0, \ldots, 1, Na_n)^{\mathrm{t}}, \\
\mathbf{v}_{n+1} &= (\tfrac{1}{2}, \tfrac{1}{2}, \ldots, \tfrac{1}{2}, Ns)^{\mathrm{t}},
\end{aligned}$$

where N is an integer larger than $\sqrt{n}$. The vector $\mathbf{v} := \epsilon_1\mathbf{v}_1 + \cdots + \epsilon_n\mathbf{v}_n - \mathbf{v}_{n+1} = (\epsilon'_1, \ldots, \epsilon'_n, 0)$ is in the lattice L, where $\epsilon'_i := \epsilon_i - \frac{1}{2} \in \{\frac{1}{2}, -\frac{1}{2}\}$. Involved calculations (carried out in Coster et al. [64, 65]) show that the probability P of the existence of a vector $\mathbf{w} \in L \setminus \{\mathbf{0}, \mathbf{v}, -\mathbf{v}\}$ with $||\mathbf{w}|| \leqslant ||\mathbf{v}||$ satisfies $P \leqslant n(4n\sqrt{n}+1)2^{cn}/B$, where $c \approx 1.0628$. Now, if the density $d(A)$ of A is less than $1/c \approx 0.9408$, then $B = 2^{c'n}$ for some $c' > c$ and, therefore, $P \to 0$ as $n \to \infty$. In other words, if $d(A) < 0.9408$, then, with a high probability, $\pm\mathbf{v}$ are the shortest non-zero vectors of L. The lattice oracle then returns such a vector from which the solution $\epsilon_1, \ldots, \epsilon_n$ can be readily computed.

4.8.2 The Lattice-Basis Reduction Algorithm

Let L be a lattice in $\mathbb{R}^n$ specified by a basis of n linearly independent vectors $\mathbf{v}_1, \ldots, \mathbf{v}_n$. We now construct a basis $\mathbf{v}^*_1, \ldots, \mathbf{v}^*_n$ of $\mathbb{R}^n$ such that $\langle \mathbf{v}^*_i, \mathbf{v}^*_j \rangle = 0$ (that is, $\mathbf{v}^*_i$ and $\mathbf{v}^*_j$ are orthogonal to each other) for all i, j, $i \neq j$. Note that $\mathbf{v}^*_1, \ldots, \mathbf{v}^*_n$ need not be a basis for L. Algorithm 4.11 is known as the *Gram–Schmidt orthogonalization* procedure.

Algorithm 4.11 **Gram–Schmidt orthogonalization**

Input: A basis $\mathbf{v}_1, \ldots, \mathbf{v}_n$ of $\mathbb{R}^n$
Output: The Gram–Schmidt orthogonalization $\mathbf{v}^*_1, \ldots, \mathbf{v}^*_n$ of $\mathbf{v}_1, \ldots, \mathbf{v}_n$.
Steps:
$\mathbf{v}^*_1 := \mathbf{v}_1$.
for $i = 2, \ldots, n$ {
  $\mathbf{v}^*_i := \mathbf{v}_i - \sum_{j=1}^{i-1} \mu_{i,j}\mathbf{v}^*_j$, where $\mu_{i,j} := \langle \mathbf{v}_i, \mathbf{v}^*_j \rangle / \langle \mathbf{v}^*_j, \mathbf{v}^*_j \rangle$.
}

One can easily verify that $\mathbf{v}^*_1, \ldots, \mathbf{v}^*_n$ constitute an orthogonal basis of $\mathbb{R}^n$. Using these notations, we introduce the following important concept:

Definition 4.9 The basis $\mathbf{v}_1, \ldots, \mathbf{v}_n$ is called a *reduced basis* of L, if

$$|\mu_{i,j}| \leqslant \tfrac{1}{2} \text{ for all } i, j,\ 1 \leqslant j < i \leqslant n, \tag{4.16}$$

and

$$||\mathbf{v}^*_i + \mu_{i,i-1}\mathbf{v}^*_{i-1}||^2 \geqslant \tfrac{3}{4}||\mathbf{v}^*_{i-1}||^2 \text{ for all } i,\ 2 \leqslant i \leqslant n\,. \tag{4.17}$$

□

A reduced basis $\mathbf{v}_1, \ldots, \mathbf{v}_n$ of L is termed so, because the vectors $\mathbf{v}_i$ are somewhat short. More precisely, we have Theorem 4.5, the proof of which is not difficult, but is involved, and is omitted here.

Theorem 4.5 Let $\mathbf{v}_1, \ldots, \mathbf{v}_n$ be a reduced basis of a lattice L, and $m \in \{1, 2, \ldots, n\}$. For any m linearly independent vectors $\mathbf{w}_1, \ldots, \mathbf{w}_m$ of L, we have

$$||\mathbf{v}_i||^2 \leqslant 2^{n-1} \max(||\mathbf{w}^2_1||, \ldots, ||\mathbf{w}_m||^2)$$

for all $i = 1, \ldots, m$. In particular, for any non-zero vector $\mathbf{w}$ of L we have

$$||\mathbf{v}_1||^2 \leqslant 2^{n-1}||\mathbf{w}||^2.$$

That is, for a reduced basis $\mathbf{v}_1, \ldots, \mathbf{v}_n$ of L the length of $\mathbf{v}_1$ is at most $2^{(n-1)/2}$ times that of the shortest non-zero vector in L. ■

Given an arbitrary basis $\mathbf{v}_1, \ldots, \mathbf{v}_n$ of a lattice L, the L^3 basis reduction algorithm computes a reduced basis of L. The algorithm starts by computing the Gram–Schmidt orthogonalization $\mathbf{v}_1^*, \ldots, \mathbf{v}_n^*$ of $\mathbf{v}_1, \ldots, \mathbf{v}_n$. The rational numbers $\mu_{i,j}$ are also available from this step. We also obtain as byproducts the numbers $V_i := \langle \mathbf{v}_i^*, \mathbf{v}_i^* \rangle$ for $i = 1, \ldots, n$.

Algorithm 4.12 enforces Condition (4.16) $|\mu_{k,l}| \leqslant 1/2$ for a given pair of indices k and l. The essential work done by this routine is subtracting a suitable multiple of $\mathbf{v}_l$ from $\mathbf{v}_k$ and updating the values $\mu_{k,1}, \ldots, \mu_{k,l}$ accordingly.

Algorithm 4.12 **Subroutine for basis reduction**

Input: Two indices k and l.
Output: An update of the basis vectors to ensure $|\mu_{k,l}| \leqslant 1/2$.
Steps:
$r := \lfloor \mu_{k,l} + \frac{1}{2} \rfloor$.
$\mathbf{v}_k := \mathbf{v}_k - r\mathbf{v}_l$.
for $h = 1, \ldots, l-1$ { $\mu_{k,h} := \mu_{k,h} - r\mu_{l,h}$. }
$\mu_{k,l} := \mu_{k,l} - r$.

If Condition (4.17) is not satisfied by some k, that is, if $V_k < (\frac{3}{4} - \mu_{k,k-1}^2)V_{k-1}$, then $\mathbf{v}_k$ and $\mathbf{v}_{k-1}$ are swapped. The necessary changes in the values V_k, V_{k-1} and certain $\mu_{i,j}$'s should also be incorporated. This is explained in Algorithm 4.13.

Algorithm 4.13 **Subroutine for basis reduction**

Input: An index k.
Output: An update of the basis vectors to ensure $V_k \geqslant (\frac{3}{4} - \mu_{k,k-1}^2)V_{k-1}$.
Steps:
$\mu := \mu_{k,k-1}$. $V := V_k + \mu^2 V_{k-1}$.
$\mu_{k,k-1} := \mu V_{k-1}/V$. $V_k := V_{k-1}V_k/V$. $V_{k-1} := V$.
Swap $(\mathbf{v}_k, \mathbf{v}_{k-1})$.
for $h = 1, \ldots, k-2$ { Swap $(\mu_{k,h}, \mu_{k,h-1})$. }
for $h = k+1, \ldots, n$ {
  $\mu' := \mu_{h,k-1} - \mu\mu_{h,k}$. $\mu_{h,k-1} := \mu_{h,k} + \mu_{k,k-1}\mu'$. $\mu_{h,k} := \mu'$.
}

The main basis reduction algorithm is described in Algorithm 4.14. It is not obvious that this algorithm should terminate at all. Consider the quantity $D := d_1 \cdots d_{n-1}$, where $d_i := |\det(\langle \mathbf{v}_k, \mathbf{v}_l \rangle)_{1 \leqslant k,l \leqslant i}|$ for each $i = 1, \ldots, n$. At the beginning of the basis reduction procedure one has $d_i \leqslant B^i$ for all $i = 1, \ldots, n$, where $B := \max(|\mathbf{v}_i|^2 \mid 1 \leqslant i \leqslant n)$. It can be shown that an invocation of Algorithm 4.12 does not alter the value of D, whereas interchanging $\mathbf{v}_i$ and $\mathbf{v}_{i-1}$ in Algorithm 4.13 decreases D by a factor $< 3/4$. It can also be shown that for any basis of L the value D is bounded from below by a constant which depends only on the lattice. Thus, Algorithm 4.14 stops after finitely many steps.

Algorithm 4.14 **Basis reduction in a lattice**

Input: A basis $\mathbf{v}_1, \ldots, \mathbf{v}_n$ of a lattice L.
Output: $\mathbf{v}_1, \ldots, \mathbf{v}_n$ converted to a reduced basis.
Steps:
Compute the Gram–Schmidt orthogonalization of $\mathbf{v}_1, \ldots, \mathbf{v}_n$ (Algorithm 4.11).
/* The initial values of $\mu_{i,j}$ and V_i are available at this point */
$i := 2$.
while $(i < n)$ {
  if $(|\mu_{i,i-1}| > 1/2)$ { Call Algorithm 4.12 with $k = i$ and $l = i - 1$. }
  if $(V_i < (\frac{3}{4} - \mu_{i,i-1}^2)V_{i-1})$ {
    Call Algorithm 4.13 with $k = i$.
    $i := \max(2, i - 1)$.
  }
  for $j = i - 2, i - 3, \ldots, 1$ {
    if $(|\mu_{i,j}| > 1/2)$ { Call Algorithm 4.12 with $k = i$ and $l = j$. }
  }
  $i{+}{+}$.
}

For a more complete treatment of the L^3 basis reduction algorithm, we refer the reader to Lenstra et al. [166] (or Mignotte [203]). It is important to note here that the L^3 basis reduction algorithm is at the heart of the Lenstra–Lenstra–Lovasz algorithm for factoring a polynomial in $\mathbb{Z}[X]$. This factoring algorithm indeed runs in time polynomially bounded by the degree of the polynomial to be factored and is one of the major breakthroughs in the history of symbolic computing.

EXERCISE SET 4.8

4.27 Let $A = \{a_1, \ldots, a_n\} \subseteq \mathbb{N}$ be a knapsack set. Show that:
(a) If A is superincreasing with $a_1 < \cdots < a_n$, then $a_i \geqslant 2^{i-1}$ for all $i = 1, \ldots, n$ and hence $d(A) \leqslant \frac{n}{n-1}$.
(b) If $d(A) \geqslant \frac{n}{\lg(2^n - 2) - \lg n}$, then there exist two different tuples $(\epsilon_1, \ldots, \epsilon_n)$ and $(\epsilon'_1, \ldots, \epsilon'_n)$ in $\{0, 1\}^n$ such that $\sum_{i=1}^n \epsilon_i a_i = \sum_{i=1}^n \epsilon'_i a_i$.

4.28 Let L be a lattice in $\mathbb{R}^n$ and let $\mathbf{v}_1, \ldots, \mathbf{v}_n$ constitute a basis of L. The *determinant* of L is defined by

$$\det L := \det(\mathbf{v}_1, \ldots, \mathbf{v}_n).$$

(a) Show that $\det L$ is an invariant of the lattice L (that is, independent of the basis $\mathbf{v}_1, \ldots, \mathbf{v}_n$ of L).
Let $\mathbf{v}_1^*, \ldots, \mathbf{v}_n^*$ be the Gram–Schmidt orthogonalization of the basis $\mathbf{v}_1, \ldots, \mathbf{v}_n$.
(b) Show that $\det L = ||\mathbf{v}_1^*|| \cdots ||\mathbf{v}_n^*||$.
(c) Prove the *Hadamard inequality*: $\det L \leqslant ||\mathbf{v}_1|| \cdots ||\mathbf{v}_n||$.

Chapter Summary

This chapter introduces the most common computationally intractable mathematical problems on which the security of public-key cryptosystems banks. We also describe some algorithms known till date for solving these difficult computational problems.

To start with, we enumerate these computational problems. The first problem in the row is the *integer factorization problem* (*IFP*) and its several variants. Some problems that are provably or believably equivalent to the IFP are the *totient problem*, problems associated with the *RSA* algorithm, and the *modular square root problem*. The next class of problems includes the *discrete logarithm problem* (*DLP*) and its variants on elliptic curves (*ECDLP*) and hyperelliptic curves (*HECDLP*). The *Diffie–Hellman problem* (*DHP*) and its variants (*ECDHP*, *HECDHP*) are believed to be equivalent to the respective variants of the DLP. Finally, the *subset sum problem* (*SSP*) and two related problems, namely the *shortest vector problem* (*SVP*) and the *closest vector problem* (*CVP*) on lattices, are introduced.

The subsequent sections are devoted to an algorithmic study of these difficult problems. We start with IFP. We first present some fully exponential algorithms like *trial division*, *Pollard's rho method*, *Pollard's* $p-1$ *method* and *Williams'* $p+1$ *method*. Next we describe the modern genre of *subexponential algorithms*. The *quadratic sieve method* (*QSM*) is discussed at length together with its heuristic improvements like *incomplete sieving*, *large prime variation* and the *multiple polynomial* variant. We also describe *TWINKLE*, a hardware device that efficiently implements the sieving stage of the QSM. We then discuss the *elliptic curve method* (*ECM*) and the *number field sieve method* (*NFSM*) for factoring integers. The NFSM turns out to be the asymptotically fastest known algorithm for factoring integers.

The (finite field) DLP is discussed next. The older *square-root methods*, such as *Shanks' baby-step–giant-step method* (*BSGS*), *Pollard's rho method* and the *Pohlig–Hellman method* (*PHM*), take exponential running times in the worst case. The PHM for a field $\mathbb{F}_q$ is, however, efficient if $q-1$ has only small prime factors. Next we discuss the modern family of algorithms collectively known as the *index calculus method* (*ICM*). For prime fields, we discuss three variants of the ICM, namely the *basic method*, the *linear sieve method* (*LSM*) and the *number field sieve method* (*NFSM*). We also discuss three variants of the ICM for fields of characteristic 2: the *basic method*, the *linear sieve method* and *Coppersmith's algorithm*. Another interesting variant is the *cubic sieve method* (*CSM*) covered in the exercises. We explain Gordon and McCurley's *polynomial sieving* in connection with Coppersmith's algorithm.

The next section deals with algorithms for solving the ECDLP. For a general elliptic curve, the exponential square-root methods are the only known algorithms. For some special classes of curves, more efficient methods are proposed in the literature. The *MOV reduction* based on *Weil pairing* reduces ECDLP on a curve over $\mathbb{F}_q$ to DLP in the finite field $\mathbb{F}_{q^k}$ for some suitable $k \in \mathbb{N}$. This k is small and the reduction is efficient for supersingular curves. The *SmartASS method* (also called the *anomalous method*) reduces the ECDLP in an anomalous curve to the computation of p-adic discrete logarithms. This reduction solves the original DLP in polynomial time. In view of these algorithms, it is preferable to avoid supersingular and anomalous curves in cryptographic applications. The *xedni calculus method* (*XCM*) is discussed finally.

This algorithm works by lifting a curve over $\mathbb{F}_p$ to a curve over $\mathbb{Q}$. Experimental and theoretical evidences suggest that the XCM is not an efficient solution to the ECDLP.

We then devote a section to the study of an index calculus method to solve the HECDLP. For hyperelliptic curves of small genus, this method leads to a subexponential algorithm (the *ADH–Gaudry algorithm*).

Many of the above subexponential methods require solving a system of linear congruences over finite rings. This (inherently sequential) linear algebra part often turns out to be the bottleneck of the algorithms. However, the fact that these equations are necessarily sparse can be effectively exploited, and some faster algorithms can be used to solve these systems. We study four such algorithms: *structured Gaussian elimination*, the *conjugate gradient method*, the *Lanczos method* and the *Wiedemann method*.

In the last section, we study the subset sum problem. We first reduce the SSP to problems associated with lattices. We finally present the *lattice-basis reduction algorithm* due to Lenstra, Lenstra and Lovasz.

Several other computationally intractable problems have been proposed in the literature for building cryptographic systems. Some of these problems are mentioned in the annotated references of Chapter 5. Due to space and time limitations, we will not discuss these problems in this book.

Suggestions for Further Reading

The integer factorization problem is one of the oldest computational problems. Though the exact notion of computational complexity took shape only after the advent of computers, the apparent difficulty of solving the factorization problem has been noticed centuries ago. Crandall and Pomerance [69] call it the fundamental computational problem of arithmetic. Numerous books and articles provide discussions on this subject at varying levels of coverage. Crandall and Pomerance [69] is perhaps the most extensive in this regard. The reader can also take a look at Bressoud's (much simpler) book [36] or the (compact, yet reasonably detailed) Chapter 10 of Henri Cohen's book [56]. The articles by Lenstra et al. [164] and by Montgomery [211] are also worth reading.

John M. Pollard has his name attached to three modern inventions in the arena of integer factorization. In [238, 239], he introduces the rho and $p-1$ methods. (Later he has been part of the team that has designed the number-field sieve factoring algorithm.) Williams' $p+1$-method appears in 1982 in [305].

The *continued fraction method* (*CFRAC*) is apparently the first known subexponential-time integer factoring algorithm. It is based on the work of Lehmer and Powers [162] and first appears in the currently used form in Morrison and Brillhart's paper [213]. CFRAC happens to be the most widely used integer factoring algorithm used during late 1970s and early 1980s.

The quadratic sieve method, invented by Carl Pomerance [241] in 1984, supersedes the CFRAC method. The multiple-polynomial QSM appears in Silverman [279]. Hendrik Lenstra's elliptic curve method [174] is proposed almost concurrently as the QSM. Nowadays, the QSM and the ECM are the most commonly used factoring methods. Reyneri's cubic sieve method is described in Lenstra and Lenstra [165].

The theoretically superior number field sieve method follows from Pollard's factoring method using cubic integers [240]. The initial proposal for the NFS method is that of the simple NFS and appears in Lenstra et al. [167]. It is later modified to the general NFS method in Buhler et al. [41]. Lenstra and Lenstra [165] is a compilation of papers on the NFS method. Though the NFS method is the asymptotically fastest factoring method, its fairly complicated implementation makes the algorithm superior to QSM or ECM, only when the bit size of the integer to be factored is reasonably large.

Shamir's factoring engine TWINKLE is proposed in [269]. A. K. Lenstra and Shamir analyse and optimize its design in [168]. Shamir and Tromer [270] have proposed a device called *TWIRL* (*The Weizmann Institute Relation Locator*) that is geared to the NFS factoring method. It is estimated that a TWIRL implementation costing US$10K can complete the sieving for a 512-bit RSA modulus in less than 10 minutes, whereas one that does the same for a 1024-bit RSA modulus costs US$10–50M and takes a time of one year. Lenstra et al. [163] provide a more detailed analysis of these estimates. See Lenstra et al. [169] to know about *Bernstein's factorization circuit* which is another implementation of the NFS factoring method.

The (finite field) discrete logarithm problem also invoked much research in the last few decades. The older square-root methods are described well in the book [191] by Menezes. Donald Knuth attributes the baby-step–giant-step method to Daniel Shanks. See Stein and Teske [290] for various optimizations of the baby-step–giant-step method. Pollard's rho method is an adaptation of the same method for integer factorization. See Pohlig and Hellman [234] for the Pohlig–Hellman method.

The first idea of the index calculus method appears in Western and Miller [302]. Coppersmith et al. [59] describe three variants of the index calculus method: the linear sieve method, the residue list sieve method and the Gaussian integer method. The same paper also proposes the cubic sieve method (CSM). LaMacchia and Odlyzko [158] describe an implementation of the linear sieve and the Gaussian integer methods. Das and Veni Madhavan [73] make an implementation study of the CSM. Also look at the survey [189] by McCurley.

Gordon [119] uses number field sieves for computing discrete logarithms over prime fields. Weber et al. [261, 299, 300, 301] have implemented and proved the practicality of the number field sieve method. Also see Schirokauer's paper [260].

Odlyzko [225] surveys the algorithms for computing discrete logs in the fields $\mathbb{F}_{2^m}$. The best algorithm for these fields is Coppersmith's algorithm [57]. No analog of this algorithm is known for prime fields. Gordon and McCurley [120] use Coppersmith's algorithm for the computation of discrete logarithms in $\mathbb{F}_{2^{401}}$ and $\mathbb{F}_{2^{503}}$.

The article [226] by Odlyzko and the one [242] by Pomerance are two recent surveys on the finite field discrete logarithm problem. Also see Buchmann and Weber [40].

The elliptic curve discrete logarithm problem seems to be a very difficult computational problem. A direct adaptation of the index calculus method is expected to lead to a running time *worse* than that of brute-force search (Silverman and Suzuki [278] and Blake et al. [24].) Menezes et al. [193] reduce the problem of computing discrete logs in an elliptic curve over $\mathbb{F}_q$ to computing discrete logs in the field $\mathbb{F}_{q^k}$ for some k. For supersingular elliptic curves, this k can be chosen to be small. For a general curve, the MOV reduction takes exponential time (Balasubramanian and Koblitz [16]). The SmartASS method is due to Smart [282], Satoh and Araki [257] and Semaev [265]. Joseph H. Silverman proposes the xedni calculus method in [277]. This method has been experimentally and heuristically shown to be impractical by Jacobson et al. [139].

Adleman et al. [2] propose the first subexponential algorithm for the hyperelliptic curve discrete log problem. This algorithm is applicable for curves of high genus over prime fields. The analysis of its running time is based on certain heuristic assumptions. Enge [86] provides a subexponential algorithm which has a rigorously provable running time and which works for curves over any arbitrary field $\mathbb{F}_q$. Again, the algorithm demands curves of high genus. An implementation of the Adleman–DeMarrais–Huang algorithm is given by Gaudry [105]. Also see Enge and Gaudry [87].

Gaudry et al. [107] propose a Weil-descent attack for the hyperelliptic curve discrete log problem. This is modified in Galbraith [100] and Galbraith et al. [101].

Coppersmith et al. [59] describe sparse system solvers. LaMacchia and Odlyzko [159] implement these methods. For further details, see Montgomery [212], Coppersmith [58], Wiedemann [303], and Yang and Brent [306].

That public-key cryptosystems can be based on the subset-sum problem (or the knapsack problem) was considered at the beginning of the era of public-key cryptography. Historically the first realization of a public-key system is based along this line and is due to Merkle and Hellman [196]. But the Merkle–Hellman system and several variants of it are broken; see Shamir [266], for example. At present, most public-key systems based on the subset-sum problem are known to be insecure.

The lattice-basis reduction algorithm and the associated L^3 algorithm for factoring polynomials appear in the celebrated work [166] of Lenstra, Lenstra and Lovasz. Mignotte's book [203] also describes these topics in good details.

Cryptographic Algorithms

5

5.1 Introduction
5.2 Secure Transmission of Messages
5.3 Key Exchange
5.4 Digital Signatures
5.5 Entity Authentication
Chapter Summary
Sugestions for Further Reading

An essential element of freedom is the right to privacy, a right that cannot be expected to stand against an unremitting technological attack.
— Whitfield Diffie

Mary had a little key (It's all she could export),
and all the email that she sent was opened at the Fort.
— Ronald L. Rivest

Treat your password like your toothbrush. Don't let anybody else use it, and get a new one every six months.
— Clifford Stoll

5.1 Introduction

As we pointed out in Chapter 1, cryptography tends to guard sensitive data from unauthorized access. We shortly describe some algorithms that achieve this goal. We restrict ourselves only to public-key algorithms. In practice, however, public-key algorithms are used in tandem with secret-key algorithms. In this chapter, we describe only the basic routines to which are input mathematical entities like integers, points in finite fields or on curves. Message encoding will be dealt with in Chapter 6.

5.2 Secure Transmission of Messages

Consider the standard scenario: a party named Alice, and called *sender*, is willing to send a secret message m to a party named Bob, and called *receiver* or *recipient*, over a public communication channel. A third party Carol may intercept and read the message. In order to maintain the secrecy of the message, Alice uses a well-defined transform f_e to convert the *plaintext message* m to the *ciphertext message* c and sends c to Bob. Bob possesses some secret information with the help of which he uses the reverse transformation f_d in order to get back m. Carol who is expected not to know the secret information cannot retrieve m from c by applying the transformation f_d.

In a public-key system, the realization of the transforms f_e and f_d is based on a *key pair* (e, d) predetermined by Bob. The *public key* e is made public, whereas the *private key* d is kept secret. The *encryption* transform generates $c = f_e(m, e)$. Since e is a public knowledge, anybody can generate c from a given m, whereas the *decryption* transform $m = f_d(c, d)$ can be performed only by Bob who possesses the knowledge of d. The key pair has to be so chosen that knowledge of e does not allow Carol to compute d in feasible time. The intractability of the computational problems discussed in Chapter 4 can be exploited to design such key pairs. The exact realization of the keys e, d and the transforms f_e, f_d depends on the choice of the underlying intractable problem and also on the way to make use of the problem. Since there are several intractable problems suitable for cryptography, there are several encryption schemes varying widely in algorithmic and mathematical details.

5.2.1 The RSA Public-key Encryption Algorithm

RSA has been the most popular encryption algorithm. Historically also, it is the first public-key encryption algorithm published in the literature (see Rivest et al. [252]). Its security is based on the intractability of the RSAP (or the RSAKIP) discussed in Exercise 4.2. Since both these problems are polynomial-time reducible to the IFP, we often say that the RSA algorithm derives its security from the intractability of the IFP. It may, however, be the case that breaking RSA is easier than factoring integers, though no concrete evidences seem to be available.

RSA key pair

Algorithm 5.1 generates a key pair for RSA.

Algorithm 5.1 **RSA key generation**

Input: A bit length l.
Output: A random RSA key pair.
Steps:
Generate two different random primes p and q each of bit length l.
$n := pq$.
Choose an integer e coprime to $\phi(n) = (p-1)(q-1)$.
$d := e^{-1} \pmod{\phi(n)}$.
Return the pair (n, e) as the public key and the pair (n, d) as the private key.

The length l of the primes p and q should be chosen large enough so as to make the factorization of n infeasible. For short-term security, values of l between 256 and 512 suffice. For long-term security, one may choose l as large as 2,048.

The random primes p and q can be generated using a probabilistic algorithm like those described in Section 3.4.2. Naive primes are normally considered to be sufficiently secure in this respect, since $p \pm 1$ and $q \pm 1$ are expected to have large prime factors in general. Gordon's algorithm (Algorithm 3.14) can also be used for generating strong primes p and q. Since Gordon's algorithm runs only nominally slower than the algorithm for generating naive primes, there is no harm in using strong primes. Safe primes, on the other hand, are difficult to generate and may be avoided.

The *RSA modulus* n is public knowledge. Determining d from n and e is easily doable, given the value of $\phi(n) = (p-1)(q-1)$ which, in turn, is readily computable, if p and q are known. If an adversary can compute $\phi(n)$ (with or without factoring n), the security of the RSA protocol based on the modulus n is compromised. However, computing $\phi(n)$ without the knowledge of p and q is (at least historically) a very difficult computational problem, and so, if n is reasonably large, RSA encryption is assumed to be sufficiently secure.

RSA encryption is done by raising the plaintext message m to the power e modulo n. In order to speed up this (modular) exponentiation, it is often expedient to take a small value for e (like 3, 257 and 65,537). However, in that case one should adopt certain precautions as Exercise 5.2 suggests. More specifically, if e entities share a common (small) encryption key e but different (pairwise coprime) moduli and if the same message m is encrypted using all these public keys, then an eavesdropper can reconstruct m easily from a knowledge of the e ciphertext messages. Another potential problem of using small e is that if m is small, that is, if $m < n^{1/e}$, then m can be retrieved by taking the *integer* e-th root of the ciphertext message.

Although the pair (n, d) is sufficient for carrying out RSA decryption, maintaining some additional (secret) information significantly speeds up decryption. To this end, it is often recommended that some or all of the values $n, e, d, p, q, d_1, d_2, h$ be stored, where $d_1 := d \operatorname{rem} (p-1)$, $d_2 := d \operatorname{rem} (q-1)$ and $h := q^{-1} \pmod{p}$.

If n can be factored, then d can be easily computed from the public key (n, e). Conversely, if n, e, d are all known, there is an efficient probabilistic algorithm which factors n. This algorithm is based on the fact that if $ed - 1 = 2^s t$ with t odd, then for at least half of the integers $a \in \mathbb{Z}_n^*$ there exists $\sigma \in \{0, \ldots, s-1\}$ such that $a^{2^\sigma t} \not\equiv \pm 1 \pmod{n}$, whereas $a^{2^{\sigma+1} t} \equiv 1 \pmod{n}$. But then the gcd of n and $a^{2^\sigma t} - 1$ is a non-trivial factor of n. For the details, solve Exercise 7.9.

Different entities in a given network should use different values of n. If two or more entities share a common n but different exponent pairs (e_i, d_i), then each entity can first factor n and then use this factorization to compute the private keys of other entities. Primes are quite abundant in nature and so finding pairwise coprime RSA moduli for all entities is no problem at all. A common value of the encryption exponent e (for example, a small value of e) can, however, be shared by all entities. In that case, for pairwise different moduli n_i, the corresponding decryption exponents d_i will also be pairwise different.

RSA encryption

RSA encryption is rather simple, as Algorithm 5.2 shows.

Algorithm 5.2 **RSA encryption**

Input: The RSA public key (n,e) of the recipient and the plaintext message $m \in \mathbb{Z}_n$.
Output: The ciphertext message $c \in \mathbb{Z}_n$.
Steps:
$c := m^e \pmod{n}$.

By Exercise 4.1, the exponentiation function $m \mapsto m^e$ is bijective; so m can be uniquely recovered from c. It is clear why small encryption exponents e speed up RSA encryption. For a general exponent e, the routine takes time $\mathrm{O}(\log^3 n)$, whereas for a small e (that is, $e = \mathrm{O}(1)$) the running time drops to $\mathrm{O}(\log^2 n)$.

RSA decryption

RSA decryption (Algorithm 5.3) is analogous to RSA encryption.

Algorithm 5.3 **RSA decryption**

Input: The RSA private key (n, d) of the recipient and the ciphertext message $c \in \mathbb{Z}_n$.
Output: The recovered plaintext message $m \in \mathbb{Z}_n$.
Steps:
$m := c^d \pmod{n}$.

The correctness of this decryption procedure follows from Exercise 4.1. As in the case of encryption, one might go for small decryption exponents d. In general, both e and d cannot be small simultaneously. If e is small, the security of the RSA scheme is expected not be affected, whereas small values of d are not desirable for several reasons. First, if d is very small, the adversary chooses some m, computes the corresponding ciphertext c (using public knowledge) and then keeps on computing $c^x \pmod{n}$ for $x = 1, 2, \ldots$ until $x = d$ is reached, that is, until the original message m is recovered.

Even when d is not very small so that the possibility of exhaustive search with $x = 1, 2, \ldots$ can be precluded, there are several attacks known for small private exponents. Wiener [304] proposes an efficient algorithm in this respect. Boneh and Durfee [32] improve Wiener's algorithm. Sun et al. [294] propose three variants of the RSA scheme that are resistant to these attacks. Durfee and Nguyen [82] extend the Boneh–Durfee attack to break two of these three variants. To sum up, it is advisable not to use small secret exponents d, that is, the bit length of d should be close to that of n in order to achieve the desired level of security.

There are alternative ways to speed up RSA decryption. If the values p, q, $d_1 := d \bmod (p-1)$, $d_2 := d \bmod (q-1)$ and $h := q^{-1} \pmod{p}$ are all available to the recipient, he can use Algorithm 5.4 for RSA decryption.

Algorithm 5.4 **RSA decryption using CRT**

Input: The RSA extended private key (p, q, d_1, d_2, h) of the recipient and the ciphertext message $c \in \mathbb{Z}_n$.
Output: The recovered plaintext message $m \in \mathbb{Z}_n$.
Steps:
$m_1 := c^{d_1} \pmod{p}$.
$m_2 := c^{d_2} \pmod{q}$.
$t := h(m_1 - m_2) \pmod{p}$.
$m := m_2 + tq$.

In this modified routine, $m_1 := m \bmod p$ and $m_2 := m \bmod q$ are first computed and then combined using the CRT to get m modulo $n = pq$. Algorithm 5.3 performs a single modular exponentiation modulo n, whereas in Algorithm 5.4 two exponentiations modulo p and q respectively take the major portion of the running time. Since an exponentiation modulo N to an exponent $\mathrm{O}(N)$ runs in time $\mathrm{O}(\log^3 N)$, and since each of p and q has bit length (about) half of that of n, Algorithm 5.4 runs about four times as fast as Algorithm 5.3.

If only the values p, q, d are stored, then d_1, d_2 and h can be computed on the fly using relatively inexpensive operations and subsequently Algorithm 5.4 can be used. This leads to a decryption routine almost as fast as Algorithm 5.4, but calls for somewhat smaller memory requirements for the storage of the private key.

5.2.2 The Rabin Public-key Encryption Algorithm

The Rabin public-key encryption algorithm is based on the intractability of computing square roots modulo a composite integer (SQRTP). By Exercise 4.10, the SQRTP is probabilistically polynomial-time equivalent to the IFP, that is, breaking the Rabin scheme is provably as hard as factoring integers. Breaking RSA, on the other hand, is only *believed* to be equivalent to factoring integers. Moreover, Rabin encryption is faster than RSA encryption (for moduli of the same size).

Rabin key pair

Like RSA, Rabin encryption requires a modulus of the form $n = pq$.

Algorithm 5.5 **Rabin key generation**

Input: A bit length l.
Output: A random Rabin key pair.
Steps:
Generate two different random primes p and q each of bit length l.
$n := pq$.
Return n as the public key and the pair (p, q) as the private key.

Here, the choice of the bit length l and the generation of the primes p and q follow the same guidelines as discussed in connection with RSA key generation.

Rabin encryption

Encryption in the Rabin scheme involves a single modular squaring.

Algorithm 5.6 **Rabin encryption**

Input: The Rabin public key n of the recipient and the plaintext message $m \in \mathbb{Z}_n$.
Output: The ciphertext message $c \in \mathbb{Z}_n$.
Steps:
$c := m^2 \pmod{n}$.

Unfortunately, the Rabin encryption map $m \mapsto m^2 \pmod{n}$ is not injective. In general, a ciphertext c has four square roots modulo n.[1] This poses ambiguity during decryption. In order to work around this difficulty, one adds some distinguishing feature or redundancy to the message m before encryption. One possibility is to duplicate a predetermined number of bits at the least significant end of m. This reduces the message space somewhat, but is rarely a serious issue. Only one of the (four) square roots of the ciphertext c is expected to have the desired redundancy. If none or more than one square root possesses the redundancy, decryption fails. However, this is a very rare phenomenon and can be ignored for all practical purposes.

Rabin decryption

Rabin decryption (Algorithm 5.7) involves computing square roots modulo n. Since n is composite, this is a very difficult problem (for the eavesdropper). But the knowledge of the prime factors p and q of n allows the recipient to decrypt.

[1] More specifically, if an element $c \in \mathbb{Z}_n$ is a square modulo both p and q, then the number of square roots of c equals 1 if $c = 0$; it is 2 if either $c \equiv 0 \pmod{p}$ or $c \equiv 0 \pmod{q}$ but not both; and it is 4 if $c \not\equiv 0 \pmod{p}$ and $c \not\equiv 0 \pmod{q}$. If c is not a square modulo either p or q, then c does not possess a square root modulo n. These assertions can be readily proved using the Chinese remainder theorem.

Algorithm 5.7 **Rabin decryption**

Input: The Rabin private key (p, q) of the recipient and the ciphertext message $c \in \mathbb{Z}_n$.
Output: The recovered plaintext message $m \in \mathbb{Z}_n$.
Steps:
if $(\left(\frac{c}{p}\right) = -1)$ or $(\left(\frac{c}{q}\right) = -1)$ { Return "c is not a ciphertext message". }
Compute the square roots of c mod p. /* Algorithm 3.17 */
Compute the square roots of c mod q. /* Algorithm 3.17 */
Compute the square roots of c mod n from those mod p and q. /* Use CRT */
if (c has exactly one distinguished square root m mod n) { Return m. }
else { Return "failure". }

5.2.3 The Goldwasser–Micali Encryption Algorithm

So far, we have encountered encryption algorithms that are *deterministic* in the sense that for a given public key of the recipient the same plaintext message encrypts to the same ciphertext message. In a *probabilistic* encryption algorithm, different calls of the encryption routine produce different ciphertext messages for the same plaintext message and public key.

The Goldwasser–Micali encryption algorithm is probabilistic and is based on the intractability of the quadratic residuosity problem (QRP) described in Exercise 4.2. If n is a composite integer and a an integer coprime to n, then $\left(\frac{a}{n}\right) = -1$ implies that a is a quadratic non-residue modulo n. The converse does not hold, that is, one may have $\left(\frac{a}{n}\right) = 1$, even when a is a quadratic non-residue modulo n. For example, if n is the product of two distinct odd primes p and q, then a is a quadratic residue modulo n if and only if a is a quadratic residue modulo both p and q. However, if $\left(\frac{a}{p}\right) = \left(\frac{a}{q}\right) = -1$, we continue to have $\left(\frac{a}{n}\right) = 1$. There is no easy way to find out if a is a quadratic residue modulo n for an integer a with $\left(\frac{a}{n}\right) = 1$. If the factorization of n is available, the QRP is solvable in polynomial time. These observations lead to the design of the Goldwasser–Micali scheme.

Goldwasser–Micali key pair

The Goldwasser–Micali scheme works in the ring $\mathbb{Z}_n$, where n is the product of two distinct sufficiently large primes. The integer a (resp. b) in Algorithm 5.8 can be found by randomly choosing elements of $\mathbb{Z}_p^*$ (resp. $\mathbb{Z}_q^*$) and computing the Legendre symbol $\left(\frac{a}{p}\right)$ (resp. $\left(\frac{b}{q}\right)$). Under the assumption that quadratic non-residues are randomly located in $\mathbb{Z}_p^*$ and $\mathbb{Z}_q^*$, a and b can be found after only a few trials. The integer x is a quadratic non-residue modulo n with $\left(\frac{x}{n}\right) = 1$.

Goldwasser–Micali encryption

Goldwasser–Micali encryption (Algorithm 5.9) is probabilistic, since its output is dependent on a sequence of random elements a_i of $\mathbb{Z}_n$. It generates a tuple $(c_1, \ldots, c_r)$

Algorithm 5.8 **Goldwasser–Micali key generation**

Input: A bit length l.
Output: A random Goldwasser–Micali key pair.
Steps:
Generate two (different) random primes p and q each of bit length l.
$n := pq$.
Find out integers a and b such that $\left(\frac{a}{p}\right) = \left(\frac{b}{q}\right) = -1$.
Compute an integer x with $x \equiv a \pmod{p}$ and $x \equiv b \pmod{q}$. /* Use CRT */
Return the pair (n, x) as the public key and the prime p as the private key.

of elements of $\mathbb{Z}_n$ such that each $\left(\frac{c_i}{n}\right) = 1$. If $m_i = 0$, then c_i is a quadratic residue modulo n, whereas if $m_i = 1$, c_i is a quadratic non-residue modulo n. Therefore, if the quadratic residuosity of c_i modulo n can be computed, the bit m_i can be determined. If one (for example, the recipient) knows the factorization of n or equivalently the prime factor p of n, one can perform decryption easily. An eavesdropper, on the other hand, must solve the QRP (or the IFP) in order to find out the bits $m_1, \ldots, m_r$. This is how Goldwasser–Micali encryption derives its security.

Algorithm 5.9 **Goldwasser–Micali encryption**

Input: The Goldwasser–Micali public key (n, x) of the recipient and the plaintext message $m = m_1 \ldots m_r$, $m_i \in \{0, 1\}$, which is a bit string of length r.
Output: The ciphertext message $(c_1, \ldots, c_r) \in \mathbb{Z}_n^r$.
Steps:
for $i = 1, \ldots, r$ {
  Select a random element $a_i \in \mathbb{Z}_n^*$.
  $c_i := x^{m_i} a_i^2$.
}

Since randomly chosen non-zero elements of $\mathbb{Z}_n$ are with high probability coprime to n, it is sufficient to draw a_i from $\mathbb{Z}_n \setminus \{0\}$ and skip the check whether $\gcd(a_i, n) = 1$. In fact, if an a_i with $\gcd(a_i, n) > 1$ is somehow located, this gcd equals a non-trivial factor of n, and the security of the scheme is broken.

The Goldwasser–Micali scheme has the drawback that the length of the ciphertext message is much bigger than that of the plaintext message. Thus, for example, for a 1024-bit modulus n and a message m of bit length 64, the output requires a huge 65,536-bit space. This phenomenon is called *message expansion* and can be a serious limitation in certain circumstances.

Goldwasser–Micali decryption

Goldwasser–Micali decryption (Algorithm 5.10) recovers the bits of the plaintext message by computing Legendre symbols modulo the prime divisor p of n. The correctness

of this decryption algorithm is evident from the discussion immediately following Algorithm 5.9.

Algorithm 5.10 **Goldwasser–Micali decryption**

Input: The Goldwasser–Micali private key p of the recipient and the ciphertext message $(c_1, \ldots, c_r) \in \mathbb{Z}_n^r$.
Output: The recovered plaintext message $m = m_1, \ldots, m_r$, $m_i \in \{0, 1\}$.
Steps:
for $i = 1, \ldots, r$ {
  if $(\left(\frac{c_i}{p}\right) = 1)$ { $m_i := 0$ } else { $m_i := 1$ }
}

5.2.4 The Blum–Goldwasser Encryption Algorithm

The Blum–Goldwasser algorithm is another probabilistic encryption algorithm and is better than the Goldwasser–Micali algorithm in the sense that in this case the message expansion is by only a constant number of bits irrespective of the length of the plaintext message. The Blum–Goldwasser scheme is based on the intractability of the SQRTP (modulo a composite integer).

Blum–Goldwasser key pair

As in the case of the encryption algorithms discussed so far, the Blum–Goldwasser algorithm works in the ring $\mathbb{Z}_n$, where $n = pq$ is the product of two distinct primes p and q. Now, we additionally demand p and q to be both congruent to 3 modulo 4.

Algorithm 5.11 **Blum–Goldwasser key generation**

Input: A bit length l.
Output: A random Blum–Goldwasser key pair.
Steps:
Generate two (different) random primes p and q each of bit length l and
  each congruent to 3 mod 4.
$n := pq$.
Return n as the public key and the pair (p, q) as the private key.

Since p and q are two different primes, there exist integers u and v such that $up + vq = 1$. In order to speed up decryption, it is often expedient to store u and v along with p and q in the private key. Recall that the solution of the congruences $x \equiv a \pmod{p}$ and $x \equiv b \pmod{q}$ is given by $x \equiv vqa + upb \pmod{n}$.

Blum–Goldwasser encryption

The Blum–Goldwasser encryption algorithm assumes that the input plaintext message m is in the form of a bit string, and breaks m into substrings of a fixed length t. A typical choice for t is $t = \lfloor \lg \lg n \rfloor$, where n is the public key of the recipient. Write $m = m_1 \dots m_r$, where each m_i is a bit string of length t. The ciphertext consists of r bit strings $c_1, \dots, c_r$, each of bit length t, and an element $d \in \mathbb{Z}_n$.

Algorithm 5.12 **Blum–Goldwasser encryption**

Input: The Blum–Goldwasser public key n of the recipient and the plaintext message $m = m_1 \dots m_r$, where each m_i is a bit string of length t.
Output: The ciphertext message $(c_1, \dots, c_r, d)$, where each c_i is a bit string of length t and $d \in \mathbb{Z}_n^*$.
Steps:
Choose a random element $d \in \mathbb{Z}_n \setminus \{0\}$.
$d := d^2 \pmod n$.
for $i = 1, \dots, r$ {
  $d := d^2 \pmod n$.
  $\delta :=$ the t least significant bits of d.
  $c_i := m_i \oplus \delta$. /* Here $\oplus$ denotes bit-wise XOR of t-bit strings */
}
$d := d^2 \pmod n$.

Blum–Goldwasser encryption involves computation of r modular squares in $\mathbb{Z}_n^*$ and is quite fast (for example, faster than RSA encryption with a general encryption exponent). It makes sense to assume that the initial choice of d is from $\mathbb{Z}_n^*$, since finding a non-zero non-invertible element of $\mathbb{Z}_n$ is as difficult as factoring n.

For an intruder to determine the plaintext message m from the corresponding ciphertext message, the values of d inside the for loop are necessary. These can be obtained by taking repeated square roots modulo n. Since n is composite, this is a difficult problem. On the other hand, since the recipient knows the prime divisors p and q of n, taking square roots modulo n requires only polynomial-time effort.

Blum–Goldwasser decryption

Recall from Exercise 3.43 that a quadratic residue $d \in \mathbb{Z}_n^*$ (where n is the public key of the recipient) has four distinct square roots of which exactly one is again a quadratic residue modulo n. This distinguished square root y of d satisfies the congruences $y \equiv d^{(p+1)/4} \pmod p$ and $y \equiv d^{(q+1)/4} \pmod q$. In the decryption Algorithm 5.13, we assume that $d \in \mathbb{Z}_n^*$.

Algorithm 5.13 assumes that each value of d is a quadratic residue modulo n. This can be verified by inserting in the for loop a check whether $\left(\frac{d}{p}\right) = \left(\frac{d}{q}\right) = 1$, before an attempt is made to compute the square root of d modulo n. If $(c_1, \dots, c_r, d)$ is a valid ciphertext message, this condition necessarily holds, and there is no fun wasting time for checking obvious things. However, if there is a possibility that d is altered by an

Algorithm 5.13 **Blum–Goldwasser decryption**

Input: The Blum–Goldwasser private key (p, q) of the recipient and the ciphertext message $(c_1, \ldots, c_r, d)$, where each c_i is a bit string of length t and $d \in \mathbb{Z}_n^*$.
Output: The recovered plaintext message $m = m_1 \ldots m_r$, where each m_i is a bit string of length t.
Steps:
for $i = r, r-1, \ldots, 1$ {
  $a := d^{(p+1)/4} \pmod p$ and $b := d^{(q+1)/4} \pmod q$.
  Compute $d \in \mathbb{Z}_n^*$ with $d \equiv a \pmod p$ and $d \equiv b \pmod q$. /* Use CRT */
  $\delta :=$ the t least significant bits of d.
  $m_i := c_i \oplus \delta$. /* XOR of t-bit strings */
}

(active) adversary (or corrupted during transmission), one may insert this check. In that case, the routine should report failure, when the square root of a quadratic non-residue modulo n is to be computed.

5.2.5 The ElGamal Public-key Encryption Algorithm

The ElGamal encryption algorithm works in a group G in which it is difficult to solve the Diffie–Hellman problem (DHP). Typical candidates for G include the multiplicative group $\mathbb{F}_q^*$ of a finite field $\mathbb{F}_q$ (usually q is a prime or a power of 2), the (additive) group of points on an elliptic curve over a finite field and the (additive) group (called the Jacobian) of reduced divisors on an hyperelliptic curve over a finite field. Here we assume that G is multiplicatively written and has order n. It is not necessary for G to be cyclic, but we should have at our disposal an element $g \in G$ with a suitably large (preferably prime) order k. We essentially work in the cyclic subgroup H of G generated by g (but using the arithmetic of G). For the ElGamal scheme, G (together with its representation), g, n and k are made public and can be shared by different entities on a network.

ElGamal key pair

Generating a key pair for the ElGamal scheme (Algorithm 5.14) involves an exponentiation in G. In order to make the exponentiation efficient, the exponent (the private key) is often chosen to have a *small* number of 1 bits. However, if this number is too small, exhaustive search by an adversary may become feasible.

If the DLP can be solved in G, the private key d can be computed from the public key g^d. This amounts to breaking a system based on this key pair. This is why we often say that the security of the ElGamal encryption scheme banks on the intractability of the DLP. But as we see shortly, the DHP is the more fundamental computational problem that dictates the security of ElGamal encryption.

Algorithm 5.14 **ElGamal key generation**

Input: G, g and k as defined above.
Output: A random ElGamal key pair.
Steps:
Generate a random integer d, $2 \leqslant d \leqslant k-1$.
Return g^d as the public key and d as the private key.

ElGamal encryption

Given a message $m \in G$, the ElGamal encryption procedure (Algorithm 5.15) generates a pair (r, s) of elements of G as the ciphertext message and thus corresponds to *message expansion* by a factor of 2. Clearly, the sender has all the relevant information for computing (r, s). The need for using a different session key for each encryption is explained in Exercise 5.6.

Algorithm 5.15 **ElGamal encryption**

Input: (G, g, k and) the ElGamal public key g^d of the recipient and the plaintext message $m \in G$.
Output: The ciphertext message $(r, s) \in H \times G$ (where $H = \langle g \rangle$).
Steps:
Generate a (random) session key d', $2 \leqslant d' \leqslant k-1$.
$r := g^{d'}$.
$s := mg^{dd'} = m(g^d)^{d'}$.

Notice that ElGamal encryption uses two exponentiations in G to exponents which are $\mathrm{O}(k)$. Therefore, the running time of Algorithm 5.15 reduces, if smaller values of k are selected. On the other hand, if k is too small, the square-root methods in $H = \langle g \rangle$ may become efficient (see Section 4.4.1). In practice, it is recommended that k be taken as a prime of length 160 bits or more.

ElGamal decryption

ElGamal decryption involves an exponentiation in G to an exponent which is $\mathrm{O}(k)$. It is easy to verify that Algorithm 5.16 performs decryption correctly and that the recipient has the necessary information to carry out decryption.

Algorithm 5.16 **ElGamal decryption**

Input: (G, g, k and) the ElGamal private key d of the recipient and the ciphertext message $(r, s) \in H \times G$ (where $H = \langle g \rangle$).
Output: The recovered plaintext message $m \in G$.
Steps:
$m := sr^{-d} = sr^{k-d}$.

An eavesdropper Carol knows the domain parameters G, g, k and n and also the recipient's public key g^d. Determining the message m from a knowledge of the corresponding ciphertext (r, s) is then equivalent to computing the element $g^{dd'}$. This implies that a (quick) solution of the DHP permits Carol to decrypt a ciphertext. If a (quick) solution of the DLP is available, then the element $g^{dd'}$ is computable fast. The reverse implication is, however, not clear: it may be easier to solve the DHP than the DLP, though no concrete evidences are available to corroborate this fact.

5.2.6 The Chor–Rivest Public-key Encryption Algorithm

The Chor–Rivest encryption algorithm is based on a variant of the subset sum problem. It selects a prime p and an integer $h \geqslant 2$, uses a knapsack set $A = \{a_0, \ldots, a_{p-1}\}$ with $1 \leqslant a_i \leqslant p^h - 2$ for each i, and considers sums of the form $s = \sum_{i=0}^{p-1} \epsilon_i a_i$, $\epsilon_i \in \{0, 1\}$, with $\sum_{i=0}^{p-1} \epsilon_i = h$. In order to construct the set A for which the h-fold sum s is uniquely determined by the binary vector $(\epsilon_0, \ldots, \epsilon_{p-1})$ of weight h (that is, with exactly h bits equal to 1), we take the help of the finite field $\mathbb{F}_{p^h}$. We represent $\mathbb{F}_{p^h}$ as $\mathbb{F}_p[X]/\langle f(X)\rangle = \mathbb{F}_p[x]$, where $f(X) \in \mathbb{F}_p[X]$ is irreducible of degree h and where x is the residue class of X in $\mathbb{F}_{p^h}$. The parameters p and h must be so chosen that $p^h - 1$ is reasonably smooth, so that the integer factorization of $p^h - 1$ can be easily computed. This helps us in two ways. First, a generator $g(x)$ of the multiplicative group $\mathbb{F}_{p^h}^*$ can be made available quickly using Algorithm 3.25. Second, the Pohlig–Hellman method of Section 4.4.1 becomes efficient for computing discrete logarithms in $\mathbb{F}_{p^h}$. We can then take $a_i := \mathrm{ind}_{g(x)}(x + i)$, $i = 0, 1, \ldots, p - 1$. If $(\epsilon_0, \ldots, \epsilon_{p-1})$ and $(\epsilon'_0, \ldots, \epsilon'_{p-1})$ are two binary vectors of weight h, then $\sum_{i=0}^{p-1} \epsilon_i a_i = \sum_{i=0}^{p-1} \epsilon'_i a_i$ implies $g(x)^{\sum_{i=0}^{p-1} \epsilon_i a_i} = g(x)^{\sum_{i=0}^{p-1} \epsilon'_i a_i}$, that is, $\prod_{i=0}^{p-1}(x + i)^{\epsilon_i} = \prod_{i=0}^{p-1}(x + i)^{\epsilon'_i}$, that is, $\epsilon_i = \epsilon'_i$ for all $i = 0, \ldots, p - 1$, since otherwise x would satisfy a non-zero polynomial of degree $< h$.

Chor–Rivest key pair

A randomly permuted version of $a_0, \ldots, a_{p-1}$ shifted by a noise (that is, a random bias) d together with p and h constitute the public key of the Chor–Rivest scheme. The private key, on the other hand, comprises the polynomials $f(X)$ and $g(x)$, the permutation just mentioned and the noise d. Algorithm 5.17 elaborates the generation of such a key pair. The same values of p and h can be used by different entities on a network. So we assume that p and h are provided instead of generated by the recipient as a part of his public key. For brevity, we use the notation $q := p^h$.

Key generation may be a long process in the Chor–Rivest scheme depending on how difficult it is to compute all the indexes $\mathrm{ind}_{g(x)}(x + i)$. Furthermore, the size of the public key is quite large, namely $O(ph \log p)$. Typically one may take $p \approx 200$ and $h \approx 25$. The original paper of Chor and Rivest [54] recommends the possibilities $(197, 24)$, $(211, 24)$, $(243, 24)$ and $(256, 25)$ for (p, h). Note that 256 is not a prime, but Chor–Rivest's algorithm works, even when p is a power of a prime. For the sake of simplicity, we here stick to the case that p is a prime.

Algorithm 5.17 **Chor–Rivest key generation**

Input: A prime p and an integer $h \geqslant 2$ such that $p^h - 1$ is smooth.
Output: A Chor–Rivest key pair.
Steps:
Choose an irreducible polynomial $f(X) \in \mathbb{F}_p[X]$ of degree h.
Use the representation $\mathbb{F}_q = \mathbb{F}_p[X]/\langle f(X)\rangle = \mathbb{F}_p[x]$, where $x := X + \langle f(X)\rangle$.
Choose a random generator $g(x)$ of $\mathbb{F}_q^*$.
Compute the indexes $a_i := \mathrm{ind}_{g(x)}(x+i)$ for $i = 0, 1, \ldots, p-1$.
Select a random permutation π of $\{0, 1, \ldots, p-1\}$.
Select a random noise d in the range $0 \leqslant d \leqslant q-2$.
Compute $\alpha_i := a_{\pi(i)} + d \pmod{q-1}$ for $i = 0, 1, \ldots, p-1$.
Return $(\alpha_0, \alpha_1 \ldots, \alpha_{p-1})$ as the public key and (f, g, π, d) as the private key.

Chor–Rivest encryption

The Chor–Rivest encryption procedure (Algorithm 5.18) assumes that the input plaintext message is represented as a binary vector $(m_0, \ldots, m_{p-1})$ of weight (that is, number of one-bits) equal to h. Since there are $\binom{p}{h}$ such binary vectors, arbitrary binary strings of bit length $\left\lfloor \lg \binom{p}{h} \right\rfloor$ can be encoded into binary vectors of the above special form. See Chor and Rivest [54] for an algorithm that describes how such an encoding can be done. Chor–Rivest encryption is quite fast, since it computes only h integer additions modulo $q-1$.

Algorithm 5.18 **Chor–Rivest encryption**

Input: The Chor–Rivest public key $(\alpha_0, \ldots, \alpha_{p-1})$ (together with p and h) and the plaintext message $(m_0, \ldots, m_{p-1})$ which is a binary vector of weight h.
Output: The ciphertext message $c \in \mathbb{Z}_{q-1}$.
Steps:
$c := \sum_{i=0}^{p-1} m_i \alpha_i \pmod{q-1}$.

Chor–Rivest decryption

The Chor–Rivest decryption procedure (Algorithm 5.19) generates a monic polynomial $v(X) \in \mathbb{F}_q[X]$ of degree h, the h (distinct) roots of which gives the non-zero bits m_i in the original plaintext message.

In order to prove that the decryption correctly works, note that $s := c - hd \equiv \sum_{i=0}^{p-1} m_i a_{\pi(i)} \pmod{q-1}$, so that $g(X)^s \equiv \prod_{i=0}^{p-1} (X + \pi(i))^{m_i} \pmod{f(X)}$. The polynomial $u(X)$ is computed as one of degree $< h$. Adding $f(X)$ to $u(X)$ gives a monic polynomial $v(X)$ of degree h, which is congruent modulo $f(X)$ to $\prod_{i=0}^{p-1}(X + \pi(i))^{m_i}$. The roots of $v(X)$ can be obtained either by a root finding algorithm or by trial divisions of $v(X)$ by $X+i$, $i = 0, 1, \ldots, p-1$. Applying the inverse of π on these roots then reconstructs the plaintext message.

Algorithm 5.19 **Chor–Rivest decryption**

Input: The Chor–Rivest private key (f, g, π, d) (together with p and h) and the ciphertext message $c \in \mathbb{Z}_{q-1}$.
Output: The recovered plaintext message $(m_0, \ldots, m_{p-1})$ which is a binary vector of weight h.
Steps:
$s := c - hd \pmod{q-1}$.
$u(X) := g(X)^s \pmod{f(X)}$.
$v(X) := f(X) + u(X)$.
Factorize $u(X)$ as $u(X) = (X + i_1) \cdots (X + i_h)$, $i_j \in \{0, 1, \ldots, p-1\}$.
For $i = 0, 1, \ldots, p-1$ set $m_i := \begin{cases} 1 & \text{if } i = \pi^{-1}(i_j) \text{ for some } j \in \{1, \ldots, h\}, \\ 0 & \text{otherwise.} \end{cases}$

An eavesdropper sees only the sum $c := \sum_{i=0}^{p-1} m_i \alpha_i \pmod{q-1}$ of the (known) knapsack weights $\alpha_0, \ldots, \alpha_{p-1}$. In order to recover $m_0, \ldots, m_{p-1}$, she should solve the SSP. By choosing p and h carefully, the density of the knapsack set can be adjusted to be high, that is, larger than what the cryptanalytic routines described in Section 4.8 can handle. Thus, the Chor–Rivest scheme is assumed to be secure. However, as discussed in Chor and Rivest [54], the security of the system breaks down, when certain partial information on the private key are available.

* 5.2.7 The XTR Public-key Encryption Algorithm

XTR, a phonetic abbreviation of *efficient and compact subgroup trace representation*, is designed by Arjen Lenstra and Eric Verheul as an attractive alternative to RSA (and similar cryptosystems including the ElGamal scheme over finite fields) and elliptic curve cryptosystems (ECC). The attractiveness of XTR arises from the following facts:

- XTR runs (about three times) faster than RSA or ECC.
- XTR has shorter keys (comparable with ECC).
- The security of XTR is based on the DLP/DHP over finite fields of sufficiently big sizes and not on a new allegedly difficult computational problem.
- The parameter and key generation for XTR is orders of magnitude faster than that for RSA/ECC.

XTR, though not a fundamental breakthrough, deserves treatment in this chapter. The working of XTR is somewhat involved and we plan to present only a conceptual description of the algorithm, hiding the mathematical details.

XTR considers the following tower of field extensions:

$$\mathbb{F}_p \subseteq \mathbb{F}_{p^2} \subseteq \mathbb{F}_{p^6},$$

where $p \equiv 2 \pmod 3$ is a prime, sufficiently large so that computing discrete logs in $\mathbb{F}_{p^6}^*$ using known algorithms is infeasible. We have $p^6 - 1 = (p-1)(p+1)(p^2 - p + 1)(p^2 + p + 1)$. Let q be a prime divisor of $p^2 - p + 1$ of bit length 160 or more.

There is a unique subgroup G of $\mathbb{F}_{p^6}^*$ with $\#G = q$. G is called the *XTR* (*sub*)*group*, whereas the entire group $\mathbb{F}_{p^6}^*$ is called the *XTR supergroup*. The XTR group G is cyclic (Lemma 2.1, p 27). Let g be a generator of G, that is, $G = \langle g \rangle = \{1, g, g^2, \ldots, g^{q-1}\}$.

The working of XTR is based on the discrete log problem in G. Since $p^2 - p + 1$ and hence q are relatively prime to the orders of the multiplicative groups of all proper subfields of $\mathbb{F}_{p^6}$, computing discrete logs in G is (seemingly) as difficult as that in $\mathbb{F}_{p^6}^*$, that is, one gets the same level of security by the use of G instead of the full XTR supergroup.

The main technical innovation of XTR is the proposal of a compact representation of the elements of G in place of the obvious representation using $\lceil 6 \lg p \rceil$ bits inherited from that of $\mathbb{F}_{p^6}$. This is precisely where the intermediate field $\mathbb{F}_{p^2}$ comes into picture. We require a map $G \to \mathbb{F}_{p^2}$, so that we can represent elements of G by those of $\mathbb{F}_{p^2}$. This map offers two benefits. First, the elements of G can now be represented using $\lceil 2 \lg p \rceil$ bits leading to a three-fold reduction in the key size. Second, the arithmetic of $\mathbb{F}_{p^2}$ can be exploited to implement the arithmetic in G, thereby improving the efficiency of encryption and decryption routines (compared to those over the full XTR supergroup).

The map $G \to \mathbb{F}_{p^2}$ uses the traces of elements of $\mathbb{F}_{p^6}$ over $\mathbb{F}_{p^2}$ (Definition 2.59). In this section, we use the shorthand notation Tr to stand for $\mathrm{Tr}_{\mathbb{F}_{p^6}|\mathbb{F}_{p^2}}$. The conjugates of an element $h \in \mathbb{F}_{p^6}$ over $\mathbb{F}_{p^2}$ are h, h^{p^2}, h^{p^4} and so

$$\mathrm{Tr}(h) = h + h^{p^2} + h^{p^4} \in \mathbb{F}_{p^2} .$$

Let us now specialize to $h = g^n \in G$, $n \in \mathbb{Z}$. Since $p^2 \equiv p - 1 \pmod{p^2 - p + 1}$ and $p^4 \equiv -p \pmod{p^2 - p + 1}$, the conjugates of h are $g^n, g^{(p-1)n}, g^{-pn}$. Thus, $\mathrm{Tr}(g^n) = g^n + g^{(p-1)n} + g^{-pn}$. Moreover,

$$\begin{aligned} & g^n g^{(p-1)n} + g^n g^{-pn} + g^{(p-1)n} g^{-pn} \\ = \; & g^{pn} + g^{(1-p)n} + g^{-n} \\ = \; & g^{pn} + g^{(1-p)n - n(p^2-p+1)} + g^{-n+n(p^2-p+1)} \\ = \; & \left(g^n + g^{-pn} + g^{(p-1)n} \right)^p = \mathrm{Tr}(g^n)^p , \end{aligned}$$

so the minimal polynomial of $h = g^n$ over $\mathbb{F}_{p^2}$ is

$$\begin{aligned} & (X - g^n)(X - g^{(p-1)n})(X - g^{-pn}) \\ = \; & X^3 - \mathrm{Tr}(g^n) X^2 + \mathrm{Tr}(g^n)^p X - 1 \in \mathbb{F}_{p^2}[X] . \end{aligned}$$

This minimal polynomial is determined uniquely by $\mathrm{Tr}(g^n)$ and so we can *represent* $g^n \in G$ by $\mathrm{Tr}(g^n) \in \mathbb{F}_{p^2}$. Note, however, that this representation is not unique, that is, the map $G \to \mathbb{F}_{p^2}$, $g^n \mapsto \mathrm{Tr}(g^n)$, is not injective. More precisely, the only elements of G that map to $\mathrm{Tr}(g^n)$ are the conjugates $g^n, g^{(p-1)n}, g^{-pn}$ of g^n. This is often not a serious problem, as we see below.

In order to complete the description of the implementation of the arithmetic of the group G, we need to address two further issues. This is necessary, since the trace representation $G \to \mathbb{F}_{p^2}$ defined above is *not* a homomorphism of groups. First, we specify how one can implement the arithmetic of $\mathbb{F}_{p^2}$. Since $p \equiv 2 \pmod 3$, $X^2 + X + 1$

is irreducible over $\mathbb{F}_p$. If $\alpha \in \bar{\mathbb{F}}_p$ is a root of $X^2 + X + 1$, we have the standard representation $\mathbb{F}_{p^2} \cong \mathbb{F}_p(\alpha) \cong \mathbb{F}_p[X]/\langle X^2 + X + 1\rangle$. That is, we can represent $\mathbb{F}_{p^2} := \{y_0 + y_1\alpha \mid y_0, y_1 \in \mathbb{F}_p\}$. Since $1 + \alpha + \alpha^2 = 0$, we have $y_0 + y_1\alpha = (-\alpha - \alpha^2)y_0 + y_1\alpha = (y_1 - y_0)\alpha + (-y_0)\alpha^2$. This leads to the non-standard representation

$$\mathbb{F}_{p^2} := \{x_1\alpha + x_2\alpha^2 \mid x_1, x_2 \in \mathbb{F}_p\}.$$

Since $p \equiv 2 \pmod 3$ and $\alpha^3 = 1 + (\alpha - 1)(\alpha^2 + \alpha + 1) = 1$, the $\mathbb{F}_p$-basis $\{\alpha, \alpha^2\}$ of $\mathbb{F}_{p^2}$ is the same as the normal basis $\{\alpha, \alpha^p\}$. Under this basis, the basic arithmetic operations in $\mathbb{F}_{p^2}$ can be implemented using only a few multiplications (and some additions/subtractions) in $\mathbb{F}_p$, as described in Table 5.1. Here, the operands are $x = x_1\alpha + x_2\alpha^2$, $y = y_1\alpha + y_2\alpha^2$ and $z = z_1\alpha + z_2\alpha^2$.

Table 5.1 Basic operations in $\mathbb{F}_{p^2}$

Operation	Number of $\mathbb{F}_p$ multiplications
x^p	0 (since $x^p = x_2\alpha + x_1\alpha^2$.)
x^2	2 (since $x^2 = x_2(x_2 - 2x_1)\alpha + x_1(x_1 - 2x_2)\alpha^2$.)
xy	3 (since $xy = (x_2y_2 - x_1y_2 - x_2y_1)\alpha + (x_1y_1 - x_1y_2 - x_2y_1)\alpha^2$, that is, it suffices to compute x_1y_1, x_2y_2, $(x_1 + x_2)(y_1 + y_2)$.)
$xz - yz^p$	4 (since $xz - yz^p = (z_1(y_1 - x_2 - y_2) + z_2(x_2 - x_1 + y_2))\alpha + (z_1(x_1 - x_2 + y_1) + z_2(y_2 - x_1 - y_1))\alpha^2$.)

Now, we explain how arithmetic operations in G translate to those in $\mathbb{F}_{p^2}$ under the representation of $g^n \in G$ by $\mathrm{Tr}(g^n) \in \mathbb{F}_{p^2}$. To start with, we show how the knowledge of $\mathrm{Tr}(h)$ and n allows one to compute $\mathrm{Tr}(h^n)$ for $h \in G$. This corresponds to an exponentiation in G. For $c \in \mathbb{F}_{p^2}$, define the polynomial

$$F_c(X) := X^3 - cX^2 + c^pX - 1 = (X - h_1)(X - h_2)(X - h_3) \in \mathbb{F}_{p^2}[X],$$

where $h_1, h_2, h_3 \in \mathbb{F}_{p^6}$ are the three roots (not necessarily distinct) of $F_c(X)$. For $n \in \mathbb{Z}$, we use the notation

$$c_n := h_1^n + h_2^n + h_3^n \in \mathbb{F}_{p^2}.$$

Putting $c = \mathrm{Tr}(g)$ yields $c_n = \mathrm{Tr}(g^n)$, or, more generally, for $c = \mathrm{Tr}(g^k)$ we have $c_n = \mathrm{Tr}(g^{kn})$. Algorithm 5.20 computes

$$S_n(c) := (c_{n-1}, c_n, c_{n+1}) \in \mathbb{F}_{p^2}^3,$$

given $c \in \mathbb{F}_{p^2}$ (for example, $\mathrm{Tr}(g^k)$) and $n \in \mathbb{Z}$ (typically $n \in \{0, 1, \ldots, q-1\}$). The correctness of the algorithm is based on the following identities, the derivations of which are left to the reader (alternatively, see Lenstra and Verheul [170]).

$$c_0 = 3 \tag{5.1}$$

$$c_1 = c \tag{5.2}$$

$$c_{-n} = c_{np} = c_n^p \tag{5.3}$$

$$c_{m+n} = c_mc_n - c_n^pc_{m-n} + c_{m-2n} \tag{5.4}$$

$$c_{n+2} = cc_{n+1} - c^p c_n + c_{n-1} \quad (5.5)$$
$$c_{2n} = c_n^2 - 2c_n^p \quad (5.6)$$
$$c_{2n-1} = c_{n-1}c_n - c^p c_n^p + c_{n+1}^p \quad (5.7)$$
$$c_{2n+1} = c_{n+1}c_n - cc_n^p + c_{n-1}^p \quad (5.8)$$

Algorithm 5.20 **XTR exponentiation**

Input: $c \in \mathbb{F}_{p^2}$ and $n \in \mathbb{Z}$.
Output: $S_n(c) = (c_{n-1}, c_n, c_{n+1}) \in \mathbb{F}_{p^2}^3$.
Steps:
if $(n < 0)$ {
  Compute $S_{-n}(c)$.
  Use Equation (5.3) to compute and return $S_n(c)$.
}
if $(n = 0)$ { Return $(c^p, 3, c)$. }
if $(n = 1)$ { Return $(3, c, c^2 - 2c^p)$. }
if $(n = 2)$ {
  Compute $S_1(c)$ and hence c_3 using Equation (5.5).
  Return (c_1, c_2, c_3).
}
/* Now $n \geqslant 3$ */
$m := \begin{cases} (n-2)/2 & \text{if } n \text{ is even,} \\ (n-1)/2 & \text{if } n \text{ is odd.} \end{cases}$ /* Let $m = (1m_{l-1}m_{l-2}\ldots m_1m_0)_2$. */
/* Initialize */
$k := 1$.
Compute $S_{2k+1}(c) = S_3(c) = (c_2, c_3, c_4)$ from $S_2(c)$ using Equation (5.5).
/* Exponentiation loop */
for $j = l-1, l-2, \ldots, 0$ {
  if $(m_j = 0)$ {
    Compute $S_{4k+1}(c) = (c_{4k}, c_{4k+1}, c_{4k+2})$ from $S_{2k+1}(c) = (c_{2k}, c_{2k+1}, c_{2k+2})$.
    /* Use Equation (5.6) for c_{4k} and c_{4k+2} and Equation (5.7) for c_{4k+1} */
  } else { /* $m_j = 1$ */
    Compute $S_{4k+3}(c) = (c_{4k+2}, c_{4k+3}, c_{4k+4})$ from $S_{2k+1}(c) = (c_{2k}, c_{2k+1}, c_{2k+2})$.
    /* Use Equation (5.6) for c_{4k+2} and c_{4k+4} and Equation (5.8) for c_{4k+3} */
  }
  $k := 2k + m_j$.
}
/* Now $k = m$ and we have computed $S_{2k+1}(c) = \begin{cases} S_{n-1}(c) & \text{if } n \text{ is even,} \\ S_n(c) & \text{if } n \text{ is odd.} \end{cases}$ */
if (n is even) {
  Compute $S_n(c) = (c_{n-1}, c_n, c_{n+1})$ from $S_{n-1}(c) = (c_{n-2}, c_{n-1}, c_n)$.
  /* Use Equation (5.5) to compute c_{n+1} from S_{n-1} */
}

A careful analysis suggests that the computation of c_n from c requires $8 \lg n$ multiplications in $\mathbb{F}_p$. An exponentiation in $\mathbb{F}_{p^6}$, on the other hand, requires an expected

number of $23.4 \lg n$ multiplications in $\mathbb{F}_p$ (assuming that, in $\mathbb{F}_{p^6}$, the time for squaring is 80 per cent of that of multiplication). Thus, the XTR representation provides a speed-up of about 3.

XTR key pair

The domain parameters for an XTR cryptosystem include primes p and q satisfying the following requirements:

- $|q| \geqslant 160$ (where $|a| = \lceil \lg a \rceil$ denotes the bit size of a positive integer a).
- $|p^6| \geqslant 1024$.
- $p \equiv 2 \pmod 3$.
- $q|(p^2 - p + 1)$.

We require a generator g of the XTR group G. Since we planned to replace working in G by working in $\mathbb{F}_{p^2}$, the element g is not needed explicitly. The trace $\mathrm{Tr}(g)$ suffices for our purpose. Lenstra and Verheul [170, 172] describe several methods for obtaining the domain parameters $p, q, \mathrm{Tr}(g)$. We describe here the naivest strategies. Algorithm 5.21 outputs the primes p, q with $|p| = l_p$ and $|q| = l_q$ for some given $l_p, l_q \in \mathbb{N}$.

Algorithm 5.21 **Generation of XTR primes**

Randomly choose $r \in \mathbb{Z}$ such that $q := r^2 - r + 1$ is a prime of size $|q| = l_q$.
Randomly choose $k \in \mathbb{Z}$ such that $p := r + kq$ is a prime with $|p| = l_p$ and $p \equiv 2 \pmod 3$.

Determination of $\mathrm{Tr}(g)$ for a suitable g requires some mathematics. First, notice that if the polynomial $F_c(X) \in \mathbb{F}_{p^2}[X]$ is irreducible (over $\mathbb{F}_{p^2}$) for some $c \in \mathbb{F}_{p^2}$, then $c = \mathrm{Tr}(h)$ for some $h \in \mathbb{F}_{p^6}^*$ with $\mathrm{ord}\, h | (p^2 - p + 1)$. Moreover, $c_{(p^2-p+1)/q}$, if not equal to 3, is the trace of an element (for example, $h^{(p^2-p+1)/q}$) of order q. Thus, we may take $\mathrm{Tr}(g) = c_{(p^2-p+1)/q}$. Although we do not need it explicitly, the corresponding $g \in G$ can be taken to be any root of the polynomial $F_{\mathrm{Tr}(g)}(X)$.

What remains to explain is how one can find an irreducible $F_c(X) \in \mathbb{F}_{p^2}[X]$. A randomized algorithm results from the fact that for a randomly chosen $c \in \mathbb{F}_{p^2}$ the polynomial $F_c(X)$ is irreducible with probability $\approx 1/3$.

Once the domain parameters of an XTR system are set, the recipient chooses a random $d \in \{2, 3, \ldots, q-2\}$ and computes $\mathrm{Tr}(g^d)$ using Algorithm 5.20. The tuple $(p, q, \mathrm{Tr}(g), \mathrm{Tr}(g^d))$ is the public key and d the private key of the recipient.

XTR encryption

XTR encryption (Algorithm 5.22) is very similar to ElGamal encryption. The only difference is that now we work in $\mathbb{F}_{p^2}$ under the trace representation of the elements of G, that is, one uses Algorithm 5.20 for computing exponentiations in G.

Algorithm 5.22 **XTR encryption**

Input: The public key $(p, q, \mathrm{Tr}(g), \mathrm{Tr}(g^d))$ of the recipient and the message $m \in \mathbb{F}_{p^2}$ to be encrypted.
Output: The ciphertext message $(r, s) \in \mathbb{F}_{p^2}^2$.
Steps:
Generate a random session key $d' \in \{2, 3, \ldots, q-2\}$.
Compute $r := \mathrm{Tr}(g^{d'})$ using Algorithm 5.20 with $c := \mathrm{Tr}(g)$ and $n := d'$.
Compute $\mathrm{Tr}(g^{dd'})$ using Algorithm 5.20 with $c := \mathrm{Tr}(g^d)$ and $n := d'$.
Set $s := m\,\mathrm{Tr}(g^{dd'})$.

XTR decryption

XTR decryption (Algorithm 5.23) is again analogous to ElGamal decryption except that we have to incorporate the XTR representation of elements of G.

Algorithm 5.23 **XTR decryption**

Input: The private key d of the recipient and the ciphertext $(r, s) \in \mathbb{F}_{p^2}^2$.
Output: The recovered plaintext message m.
Steps:
Compute $\mathrm{Tr}(g^{dd'})$ using Algorithm 5.20 with $c := r = \mathrm{Tr}(g^{d'})$ and $n := d$.
Set $m := s\,\mathrm{Tr}(g^{dd'})^{-1} \in \mathbb{F}_{p^2}$.

Note that XTR encryption and decryption use Algorithm 5.20 for performing exponentiations. Therefore, these routines run about three times faster than the corresponding ElGamal routines based on the standard $\mathbb{F}_{p^6}$ arithmetic.

* 5.2.8 The NTRU Public-key Encryption Algorithm

Hoffstein et al. [130] have proposed the NTRU encryption scheme in which encryption involves a mixing system using the polynomial algebra $R := \mathbb{Z}[X]/\langle X^n - 1\rangle$ and reductions modulo two relatively prime integers α and β. The decryption involves an unmixing system and can be proved to be correct with high probability. The security of this scheme banks on the interaction of the mixing system with the independence of the reductions modulo α and β. Attacks against NTRU based on the determination of short vectors in certain lattices are known. However, suitable choices of the parameters make NTRU resistant to these attacks. The most attractive feature of the NTRU scheme is that encryption and decryption in this case are much faster than those in other known schemes (like RSA, ECC and even XTR).

NTRU key pair

NTRU parameters include three positive integers n, α and β with $\gcd(\alpha, \beta) = 1$ and with β considerably larger than α (see Table 5.2). Consider the polynomial algebra

$R := \mathbb{Z}[X]/\langle X^n - 1\rangle$. An element of $f \in R$ is represented as a polynomial $f = f_0 + f_1 X + \cdots + f_{n-1}X^{n-1}$ or, equivalently, as a vector $(f_0, f_1, \ldots, f_{n-1})$ of the coefficients. Note that $X^n - 1$ is not irreducible in $\mathbb{Z}[X]$ (for $n \geqslant 2$) and so R is not a field, but that does not matter for the NTRU scheme. For two polynomials f, g of degree $< n$ and with integer coefficients, we denote by fg the product of f and g in $\mathbb{Z}[X]$, whereas f and g as elements of R multiplies to $f \circledast g = h$ with

$$h_i = \sum_{j=0}^{i} f_j g_{i-j} + \sum_{j=i+1}^{n-1} f_j g_{n+i-j} = \sum_{\substack{0 \leqslant j,k \leqslant n-1 \\ j+k \equiv i \pmod n}} f_j g_k \,, \text{ for } i = 0, 1, \ldots, n-1.$$

NTRU works with polynomials having small coefficients. More specifically, we define the following subsets of R. The message space (that is, the set of plaintext messages) $\mathcal{R}_m$ consists of all polynomials of R with coefficients reduced modulo α. Unlike our representation of $\mathbb{Z}_\alpha$ so far, we use the integers between $-\alpha/2$ and $+\alpha/2$ to represent the coefficients of polynomials in $\mathcal{R}_m$, that is,

$$\mathcal{R}_m := \{f \in R \mid \text{ the coefficients of } f \text{ are } > -\alpha/2 \text{ and } \leqslant \alpha/2\}\,.$$

For $\nu_1, \nu_2 \in \mathbb{N}_0$, we also define the subset

$$\mathcal{R}(\nu_1, \nu_2) := \{f \in R \mid \ \nu_1 \text{ (resp. } \nu_2\text{) coefficients of } f \text{ are } 1 \text{ (resp. } -1\text{), the rest } 0\}$$

of R. For suitably chosen parameters ν_f, ν_g and ν_u (see Table 5.2), we use the special notations:

$$\mathcal{R}_f := \mathcal{R}(\nu_f, \nu_f - 1), \quad \mathcal{R}_g := \mathcal{R}(\nu_g, \nu_g), \quad \mathcal{R}_u := \mathcal{R}(\nu_u, \nu_u)\,.$$

With these notations we are now ready to describe the NTRU key generation routine. The subsets $\mathcal{R}_m$, $\mathcal{R}_f$, $\mathcal{R}_g$ and $\mathcal{R}_u$ are assumed to be public knowledge (along with the parameters n, α and β).

Algorithm 5.24 **NTRU key generation**

Input: n, α, β and $\mathcal{R}_f, \mathcal{R}_g$ as defined above.
Output: A random NTRU key pair.
Steps:
Choose $f \in \mathcal{R}_f$ and $g \in \mathcal{R}_g$ randomly.
/* f must be invertible modulo both α and β */
Compute f_α and f_β satisfying $f_\alpha \circledast f \equiv 1 \pmod \alpha$ and $f_\beta \circledast f \equiv 1 \pmod \beta$.
$h := f_\beta \circledast g \pmod \beta$.
Return h as the public key and f (along with f_α) as the private key.

The polynomial f_α can be computed from f during decryption. However, for the sake of efficiency, it is recommended that f_α be stored along with f.

The integers α and β are either small primes or small powers of small primes (Table 5.2). The most time-consuming step in the NTRU key generation procedure is the computation of the inverses f_α and f_β. Suppose we want to compute the inverse of

Table 5.2 Recommended NTRU parameters

Security	n	α	β	ν_f	ν_g	ν_u
short-term	107	3	64	15	12	5
moderate	167	3	128	61	20	18
standard*	263	3	128	50	24	16
high	503	3	256	216	72	55

*Assumed to be equivalent to 1024-bit RSA

f in $(\mathbb{Z}/p^e\mathbb{Z})[X]/\langle X^n-1\rangle$, where p is a small prime and e is a small exponent (we may have $e=1$). We first compute $f(X)^{-1}$ in the ring $(\mathbb{Z}/p\mathbb{Z})[X]/\langle X^n-1\rangle$. Since p is a prime, $\mathbb{Z}/p\mathbb{Z}=\mathbb{F}_p$ is a field, that is, $(\mathbb{Z}/p\mathbb{Z})[X]$ is a Euclidean domain (Exercise 2.31). We compute the extended Euclidean gcd of $f(X)$ with X^n-1. If $f(X)$ and X^n-1 are not coprime modulo p, then $f(X)$ is not invertible in $(\mathbb{Z}/p\mathbb{Z})[X]/\langle X^n-1\rangle$, else we get $1=s(X)f(X)+t(X)(X^n-1)\in(\mathbb{Z}/p\mathbb{Z})[X]$ and $s(X)$ is the inverse of $f(X)$ in $(\mathbb{Z}/p\mathbb{Z})[X]/\langle X^n-1\rangle$. A randomly chosen $f(X)$ with $\gcd(f(1),p)=1$ has high probability of being invertible modulo p. Recall that we have chosen $f\in\mathcal{R}_f=\mathcal{R}(\nu_f,\nu_f-1)$, so that $f(1)=1$.

If $e=1$, we have already computed the desired inverse of $f(X)$. If $e>1$, we have to *lift* the inverse $f_p(X)=u(X)$ of $f(X)$ modulo p to the inverse $f_{p^2}(X)$ of $f(X)$ modulo p^2, and then to the inverse $f_{p^3}(X)$ of $f(X)$ modulo p^3, and so on. Eventually, we get the inverse $f_{p^e}(X)$ of $f(X)$ modulo p^e. Here we describe the generic lift procedure of $f_{p^k}(X)$ to $f_{p^{k+1}}(X)$. In the ring $R=\mathbb{Z}[X]/\langle X^n-1\rangle$, we have $f_{p^k}\circledast f\equiv 1 \pmod{p^k}$. We can write $f_{p^{k+1}}(X)=f_{p^k}(X)+p^k a(X)$ for some $a(X)\in R$. Substituting this value in $f_{p^{k+1}}\circledast f\equiv 1 \pmod{p^{k+1}}$ gives the unknown polynomial $a(X)$ as

$$a(X)\equiv s(X)\circledast\left(\frac{1-f_{p^k}\circledast f}{p^k}\right) \pmod p,$$

where $s(X)=f_p(X)$ is the inverse of f modulo p.

It is often recommended that $f(X)$ be taken of the form $f(X)=1+\alpha\hat{f}(X)$ for some $\hat{f}(X)\in R$. In this case, $f_\alpha(X)=1$ is trivially available and need not be computed as mentioned above. Such a choice of f also speeds up NTRU decryption (see Algorithm 5.26) by reducing the number of polynomial multiplications from two to one. The inverse f_β, however, has to be computed (but need not be stored).

NTRU encryption

For NTRU encryption (Algorithm 5.25), the message is encoded to a polynomial in $\mathcal{R}_m$. The costliest step in this algorithm is computing the product $u\circledast h$ and can be done in time $O(n^2)$. Asymptotically better running time ($O(n\log n)$) is achievable by Algorithm 5.25, if one uses faster polynomial multiplication routines (like those based on fast Fourier transforms). However, for the cryptographic range of values of n, straightforward quadratic multiplication gives better performance. Most other encryption schemes (like RSA) take time $O(n^3)$, where n is the size of the modulus. This explains why NTRU encryption is much faster than conventional encryption routines.

Algorithm 5.25 **NTRU encryption**

Input: (n, α, β and) the NTRU public key h of the recipient and the plaintext message $m \in \mathcal{R}_m$.
Output: The ciphertext c which is a polynomial in R, reduced modulo β.
Steps:
Randomly select $u \in \mathcal{R}_u$.
$c := \alpha u \circledast h + m \pmod{\beta}$.

NTRU decryption

NTRU decryption (Algorithm 5.26) involves two multiplications in R and runs in time $\mathrm{O}(n^2)$. In order to prove the correctness of Algorithm 5.26, one needs to verify that $v \equiv \alpha u \circledast g + f \circledast m \pmod{\beta}$. With an appropriate choice of the parameters, it can be ensured that almost always the polynomial $\alpha u \circledast g + f \circledast m \in R$ has coefficients in the interval $-\beta/2$ and $+\beta/2$. In that case, we have the equality $v = \alpha u \circledast g + f \circledast m$ in R. Multiplication of v by f_α and reduction modulo α now clearly retrieves m.

Algorithm 5.26 **NTRU decryption**

Input: The NTRU private key f (and f_α) of the recipient and the ciphertext message c.
Output: The recovered plaintext message $m \in \mathcal{R}_m$.
Steps:
$v := f \circledast c \pmod{\beta}$.
/* The coefficients of v are chosen to lie between $-\beta/2$ and $+\beta/2$ */
$m := f_\alpha \circledast v \pmod{\alpha}$.

If f is chosen to be of the special form $f = 1 + \alpha f_1$ (for some polynomial f_1), then $v = \alpha u \circledast g + \alpha f_1 \circledast m + m$. Thus, reduction of v modulo α straightaway gives m, that is, there is no need to multiply v by f_α. Also f_α (having the trivial value 1) need not be stored in the private key. To sum up, taking f to be of the above special form increases the efficiency of the NTRU scheme without (seemingly) affecting its security. But now f is no longer an element of $\mathcal{R}_f$ and some care should be taken to choose suitable values of f.

NTRU decryption fails, usually when m is not properly centred (around 0). In that case, representing v as a polynomial with coefficients in the range $-\beta/2 + x$ and $+\beta/2 + x$ for a small positive or negative value of x may result in correct decryption. If, on the other hand, no values of x work, NTRU decryption cannot recover m easily and is said to suffer from a *gap failure*. For suitable parameter values, gap failures are very unlikely and can be ignored for all practical purposes.

Now, let us see how the NTRU system can be broken. In order to find out the private key f from the public key $h = f_\beta \circledast g$, one may keep on searching for $f \in \mathcal{R}_f$ exhaustively, until $f \circledast h \pmod{\beta} \in \mathcal{R}_g$. Alternatively, one may try all $g \in \mathcal{R}_g$, until $g \circledast h^{-1} \pmod{\beta} \in \mathcal{R}_f$. In a similar manner, m can be retrieved from c by trying all

$u \in \mathcal{R}_u$, until $c - \alpha u \circledast h \pmod{\beta} \in \mathcal{R}_m$. Clearly, such an attack takes expected time proportional to the size of $\mathcal{R}_f$ or $\mathcal{R}_g$ or $\mathcal{R}_u$.

A baby-step–giant-step strategy reduces the running times to the square roots of the sizes of the above sets. For example, suppose we want to compute f from h. We split $f = f_1 + f_2$ into two nearly equal pieces f_1 and f_2. If n is odd, f_1 may contain the $(n+1)/2$ most significant terms and f_2 the $(n-1)/2$ least significant terms of f. Now, we compute $(f_2, -f_2 \circledast h \pmod{\beta})$ for all possibilities of f_2 and store the pairs sorted by the second component. Next, for each possibility of f_1 (baby step) we compute $f_1 \circledast h \pmod{\beta}$ and see if there is any f_2 (giant step) for which $f_1 \circledast h \pmod{\beta}$ and $-f_2 \circledast h \pmod{\beta}$ have nearly equal values. If a matching pair (f_1, f_2) is located, we take $f = f_1 + f_2$. A similar method works for guessing m from c.

It is necessary to take the sets $\mathcal{R}_f$, $\mathcal{R}_g$ and $\mathcal{R}_u$ big enough, so that exhaustive or square root attacks are not feasible. Typically, choosing the sizes of these sets to be $\geqslant 2^{160}$ is deemed sufficiently secure.

Another relevant attack is discussed in Exercise 5.11. By far, the most sophisticated attack on the NTRU encryption scheme is based on finding short vectors in a lattice. We describe this attack in connection with the computation of the private key f from a knowledge of the private key h. Let L denote the lattice in $\mathbb{R}^{2n}$ generated by the rows of the $2n \times 2n$ matrix:

$$\begin{pmatrix}
\lambda & 0 & \cdots & 0 & h_0 & h_1 & \cdots & h_{n-1} \\
0 & \lambda & \cdots & 0 & h_{n-1} & h_0 & \cdots & h_{n-2} \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & \lambda & h_1 & h_2 & \cdots & h_0 \\
0 & 0 & \cdots & 0 & \beta & 0 & \cdots & 0 \\
0 & 0 & \cdots & 0 & 0 & \beta & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & 0 & 0 & 0 & \cdots & \beta
\end{pmatrix},$$

where $h = h_0 + h_1 X + \cdots + h_{n-1} X^{n-1} = (h_0, h_1, \ldots, h_{n-1})$ and where λ is a parameter whose choice is discussed below. Since $h = g \circledast f^{-1} \pmod{\beta}$, multiplying the i-th row by f_{i-1} ($i = 1, \ldots, n$) and adding we conclude that the vector $\mathbf{v} := (\lambda f_0, \lambda f_1, \ldots, \lambda f_{n-1}, g_0, g_1, \ldots, g_{n-1})$ is in L. By tuning the value λ, the attacker maximizes the chance for $\mathbf{v}$ to be a short vector in L. However, if the system parameters are appropriately selected, lattice reduction algorithms become rather ineffective in finding $\mathbf{v}$. Heuristic evidences suggest that this attack runs in time exponential in n.

EXERCISE SET 5.2

5.1 Establish the correctness of Algorithm 5.4.

5.2 **(a)** Assume that the same message m is encrypted using the RSA algorithm and using the public keys $(n_1, e), \ldots, (n_e, e)$ of e entities each of which has the same encryption exponent e. Assume further that the moduli $n_1, \ldots, n_e$ are pairwise coprime. Specify a method by which an adversary can reconstruct the message m from a knowledge of the ciphertext messages $c_1, \ldots, c_e$. **[H]**

(b) How can such an attack be prevented? **[H]**

5.3 **(a)** Let $n, e \in \mathbb{N} \setminus \{1\}$. How many solutions does the polynomial $X^e - X$ have in $\mathbb{Z}_n^*$? **[H]**
(b) In particular, conclude that if $n = pq$ is an RSA modulus and e is the encryption exponent, there exist $\gcd(e-1, p-1) \times \gcd(e-1, q-1)$ messages m for which $m^e \equiv m \pmod{n}$. Such messages are often called *unconcealed*. The number of unconcealed messages for random parameters n and e is, in general, vanishingly low compared to n.

5.4 Assume that two parties Bob and Barbara share a common RSA modulus n but relatively prime encryption exponents e_1 and e_2. Alice encrypts the same message by (n, e_1) and (n, e_2) and sends the ciphertext messages to Bob and Barbara respectively. Suppose also that Carol intercepts both the ciphertexts. Describe a method by which Carol retrieves the (common) plaintext. **[H]**

5.5 Let $n = pq$ be a Rabin public key and let $c \in \mathbb{Z}_n^*$ be a quadratic residue modulo n. Show that the knowledge of the four square roots of c modulo n breaks the Rabin system.

5.6 What is the disadvantage of using the same session key in the ElGamal encryption scheme for encrypting two different messages (for the same recipient)? **[H]**

5.7 Let p be an odd prime and g a generator of $\mathbb{Z}_p^*$.
(a) Show that the set $S := \{g^{2i} \mid i = 0, 1, \ldots, (p-3)/2\}$ is precisely the set of all quadratic residues modulo p. Show also that S is a subgroup of $\mathbb{Z}_p^*$.
(b) Assume that $y \equiv g^x \pmod{p}$ for some $x \in \{0, 1, \ldots, p-2\}$. Show that the least significant bit of x is 0 or 1 according as whether $y^{(p-1)/2}$ is congruent to 1 or -1 modulo p respectively. Thus, it is *easy* to determine from y the least significant bit of the discrete logarithm $x = \operatorname{ind}_g y$.
(c) Assume that $p \equiv 3 \pmod{4}$ and that p, g, y are only known (but x is not known). Suppose further that there is an oracle (a black box) that, given $z \in \mathbb{Z}_p^*$, returns the second least significant bit of $\operatorname{ind}_g z$. Show that $x = \operatorname{ind}_g y$ can be easily computed by making a polynomial (in $\log p$) number of calls to this oracle. **[H]**

5.8 Show that if the private-key parameters $f(X)$ and d are known to a cryptanalyst of the Chor–Rivest scheme, she can recover the other parts of the private key and thus break the system completely. **[H]**

5.9 Show that if $f(X)$ is only known to a cryptanalyst of the Chor–Rivest scheme, then also she can recover the full private key. **[H]**

5.10 **(a)** Derive the identities of Equations (5.1) through (5.8) (p 325).
(b) With the notations of Section 5.2.7 deduce that:

$$\begin{aligned} c_3 &= c^3 - 3c^{p+1} + 3\,. \\ c_4 &= c^4 - 4c^{p+2} + 2c^{2p} + 4c\,. \end{aligned}$$

5.11 In this exercise, we use the notations of Section 5.2.8. Assume that Alice encrypts the same message m several times using the NTRU public key h of Bob, but with different random polynomials $u_i \in \mathcal{R}_u$, $i = 1, \ldots, r$, and sends the corresponding ciphertext messages $c_1, \ldots, c_r$. Describe a strategy how an eavesdropper Carol can recover a considerable part of u_1. **[H]** Trying all the possibilities for the (relatively small) unknown part of u_1 allows Carol to retrieve m with little effort.

5.3 Key Exchange

Consider the scenario wherein two parties Alice and Bob want to share a secret information (say, a DES key for future correspondence), but it is not possible to communicate this secret by personal contact or by conversing over a secure channel. In other words, Alice and Bob want to arrive at a common secret value by communicating over a public (and hence insecure) channel. A *key-exchange* or a *key-agreement* protocol allows Alice and Bob to do so. The protocol should be such that an eavesdropper listening to the conversation between Alice and Bob cannot compute the secret value in feasible time.

Public-key technology is used to design a key-exchange protocol in the following way. Alice generates a key pair (e_A, d_A) and sends the public key e_A to Bob. Similarly, Bob generates a random key pair (e_B, d_B) and sends the public key e_B to Alice. Now, Alice and Bob respectively compute the values $s_A = f(e_B, d_A)$ and $s_B = f(e_A, d_B)$ using their respective knowledges, where f is a suitably chosen function. If $s_A = s_B$, then this value can be used as the shared secret between Alice and Bob. The intruder Carol can intercept e_A and e_B, but f should be such that a knowledge of e_A and e_B alone does not allow Carol to compute $s_A = s_B$. She needs d_A or d_B for this computation. Since (e_A, d_A) and (e_B, d_B) are key pairs, we assume that it is infeasible to compute d_A from e_A or d_B from e_B.

In what follows, we describe some key-exchange protocols. The security of these protocols is dependent on the intractability of the DHP (or the DLP). We provide a generic description, where we work in a finite Abelian multiplicative group G of order n. We write the identity of G as 1. G need not be cyclic, but we assume that an element $g \in G$ having suitably large (and preferably prime) multiplicative order m is provided. G, g, n and m may be made publicly available, but G should be a group in which one cannot compute discrete logarithms in feasible time. Typical examples of G are given in Section 5.2.5.

5.3.1 Basic Key-Exchange Protocols

Basic key-exchange protocols provide provable security against passive attacks under the intractability of the DHP. However, several models of active attacks are known for the basic protocols. One requires authentication (validation of the public keys) to eliminate these attacks.

The Diffie–Hellman key-exchange protocol

The Diffie–Hellman (DH) key-exchange algorithm [78] is one of the pioneering discoveries leading to the birth of public-key cryptography.

Algorithm 5.27 **Diffie–Hellman key exchange**

Input: G, g, n and m as defined above.
Output: A secret element $s \in G$ to be shared by Alice and Bob.
Steps:
Alice generates a random $d_A \in \{2, \ldots, m-1\}$ and computes $e_A := g^{d_A}$.
Alice sends e_A to Bob.
Bob generates a random $d_B \in \{2, \ldots, m-1\}$ and computes $e_B := g^{d_B}$.
Bob sends e_B to Alice.
Alice computes $s := (e_B)^{d_A} = g^{d_A d_B}$.
Bob computes $s := (e_A)^{d_B} = g^{d_A d_B}$.
if $(s = 1)$ { Return "failure". }

The DH scheme fails, if the shared secret turns out to be a trivial element (like the identity) of G. In that case, Alice and Bob should re-execute the protocol with different key pairs. The probability of such an incident is, however, extremely low.

The intruder Carol learns the group elements g^{d_A} and g^{d_B} by listening to the conversation between Alice and Bob and intends to compute $s = g^{d_A d_B}$. Thus, she has to solve an instance of the DHP in the group G. By assumption, this is computationally infeasible. This is how the DH scheme derives its security.

Small-subgroup attacks

A *small-subgroup attack* on the DH protocol can be mounted by an active adversary. Assume that the order m of g in G is composite and has known factorization $m = uv$ with u small. Carol intercepts the messages between Alice and Bob, replaces them by their respective v-th powers and retransmits the modified messages.

Algorithm 5.28 **A small-subgroup attack by an active eavesdropper**

Alice generates a random $d_A \in \{2, \ldots, m-1\}$ and computes $e_A := g^{d_A}$.
Alice transmits e_A for Bob.
Carol intercepts e_A, computes $e'_A := (e_A)^v$ and sends e'_A to Bob.
Bob generates a random $d_B \in \{2, \ldots, m-1\}$ and computes $e_B := g^{d_B}$.
Bob transmits e_B for Alice.
Carol intercepts e_B, computes $e'_B := (e_B)^v$ and sends e'_B to Alice.
Alice computes $s' := (e'_B)^{d_A} = g^{v d_A d_B}$.
Bob computes $s' := (e'_A)^{d_B} = g^{v d_A d_B}$.
if $(s' = 1)$ { Return "failure". }

But $\operatorname{ord} g = uv$ and so $(s')^u = 1$, that is, s' has only $u - 1$ non-trivial values. Since u is small, the possibilities for s' can be exhaustively searched by Carol. The best countermeasure against this attack is to take m to be a prime (of bit length $\geqslant 160$).

Even when m is prime, it may be the case that the cofactor $k := n/m$ has a small divisor u and it is possible that an active attacker intervenes in such a way that Alice and Bob agree upon a secret value of order (equal to or dividing) u. For example,

Carol may replace both the transmitted public keys by an element h of order u. If d_A and d_B are congruent modulo u, the shared secret has only a few possible values and Carol can obtain the correct value by exhaustive search. On the other hand, if $d_A \not\equiv d_B \pmod{u}$, Alice and Bob do not come up with the same secret. However, if Alice uses her secret to encrypt a message for Bob, it remains easy for Carol to decrypt the intercepted ciphertext by trying only a few choices for Alice's key. Alice and Bob can prevent this attack by refusing to accept as the shared secret not only the trivial value $s = 1$ but also elements of small orders.

A small-subgroup attack can also be mounted by one of the communicating parties (say, Bob) in an attempt to gain information about the other's (Alice's) secret d_A. Let us continue to assume that the cofactor $k := n/m$ has a small divisor u. Bob finds an element h in G of order u. Instead of $e_B = g^{d_B}$ Bob now sends $\hat{e}_B = g^{d_B} h$ to Alice. Alice computes the shared secret as $\hat{s}_A = (\hat{e}_B)^{d_A} = g^{d_A d_B} h^{d_A}$. Bob, on the other hand, can normally compute $s_B := g^{d_A d_B}$. Now, suppose that Alice uses a symmetric cipher with the key $\hat{s}_A$ (or some part of it) and sends the ciphertext to Bob. In order to decrypt, Bob tries all of the u possible keys $s_B h^j$ for $j = 0, 1, \ldots, u-1$. The value of j for which decryption succeeds equals d_A modulo u. A similar attack can be mounted by Bob, when $\hat{e}_B$ is chosen to be an element (like h itself) of order u.

If G is cyclic and H is the subgroup generated by g, then an element $a \in G$ is in H if and only if $a^m = 1$ (Proposition 2.5, p 27). Moreover, if $\gcd(k, m) = 1$, each communicating party can check the validity of the other party's public key by using an m-th power exponentiation. An element like $\hat{e}_B = g^{d_B} h$ or h of the last paragraph does not pass this test. If so, Alice should abandon the protocol. However, the validation of the public key requires a modular exponentiation and thereby slows down the protocol.

Cofactor exponentiation

We now present an efficient modification of the basic Diffie–Hellman scheme that prevents small-subgroup attacks (by a communicating party or an eavesdropper) without calculating an extra exponentiation. We continue with the notation $k := n/m$ and assume that k is coprime to m. Now, the shared secret is computed as $g^{d_A d_B}$ or $g^{k d_A d_B}$ depending on whether compatibility with the original DH scheme is desired or not. Algorithm 5.29 describes the modified DH algorithm. Solve Exercise 5.12 in order to establish the effectiveness of this algorithm against small-subgroup attacks.

5.3.2 Authenticated Key-Exchange Protocols

Other active attack models on the (basic or modified) DH protocol can be conceived of. One important class of attacks is now described.

Unknown key-share attacks

An *unknown key-share attack* on a key-exchange protocol makes a party believe that (s)he shares a secret with another party, whereas the secret is actually shared by a third party. Assume that Carol can monitor and modify every message between Alice and

Algorithm 5.29 **Diffie–Hellman key exchange with cofactor exponentiation**

Input: G, g, n, m and k as defined above and a flag indicating compatibility with the original DH scheme.
Output: A secret element $s \in G$ to be shared by Alice and Bob.
Steps:
Alice generates a random $d_A \in \{2, \ldots, m-1\}$ and computes $e_A := g^{d_A}$.
Alice sends e_A to Bob.
Bob generates a random $d_B \in \{2, \ldots, m-1\}$ and computes $e_B := g^{d_B}$.
Bob sends e_B to Alice.
if (compatibility with the original DH algorithm is desired) {
  Alice assigns $\delta_A := k^{-1} d_A \pmod{m}$.
  Bob assigns $\delta_B := k^{-1} d_B \pmod{m}$.
} else {
  Alice assigns $\delta_A := d_A \pmod{m}$.
  Bob assigns $\delta_B := d_B \pmod{m}$.
}
Alice computes $s := (e_B)^{k\delta_A}$.
Bob computes $s := (e_A)^{k\delta_B}$.
if $(s = 1)$ { Return "failure". }

Bob. When Alice and Bob execute Algorithm 5.27 or 5.29, Carol can intervene and pretend to Alice that she is Bob and to Bob that she is Alice. At the end of the protocol, Alice and Carol come up with a shared secret s_{AC}, and Bob and Carol with another shared secret s_{BC}. Alice believes that she shares s_{AC} with Bob, and Bob believes that he shares s_{BC} with Alice.

Now, when Alice wants to send a secret message m to Bob, she encrypts m by s_{AC} and transmits the ciphertext c. Carol intercepts c, decrypts it by s_{AC} to retrieve m, encrypts m by s_{BC} and sends the new ciphertext c' to Bob. Bob retrieves m by decrypting c' with his key s_{BC}. The process raises hardly any suspicion in Alice or Bob about the existence of the mediating third party.

In order to avoid this attack, Alice and Bob should each validate the authenticity of the public key of the other party. Public-key certificates can be used to this effect. Unfortunately, using certificates alone may fail to eliminate unknown key-share attacks, as Algorithm 5.30 shows. At the end of this protocol Alice and Bob share a secret s, but Bob *believes* that he shares it with (the intruder) Carol. Here Carol herself cannot compute the shared secret s (provided that computing discrete logs in G is infeasible). Still there may be situations where this attack can be exploited (see Law et al. [161] for a hypothetical example).

This attack has two potential problems. Under the assumption of intractability of the DLP in G, Carol cannot compute the private key corresponding to the public key e_C and so her getting the certificate $Cert_C$ knowing e_C alone may be questioned. Furthermore, replacing $(e_B, Cert_B)$ to $((e_B)^d, Cert_B)$ may make the certificate invalid. If we assume that a certificate authenticates only the entity and not the public key, then these objections can be overruled. In practice, however, a public key certificate should bind

Algorithm 5.30 **An unknown key-share attack**

Alice generates a random $d_A \in \{2, \ldots, m-1\}$ and computes $e_A := g^{d_A}$.
Alice gets the certificate $Cert_A$ on e_A from the certifying authority.
Alice transmits $(e_A, Cert_A)$ for Bob.
Carol intercepts $(e_A, Cert_A)$.
Carol chooses a random $d \in \{1, 2, \ldots, m-1\}$.
Carol gets the certificate $Cert_C$ on $e_C := (e_A)^d$ from the certifying authority.
Carol sends $(e_C, Cert_C)$ to Bob.
Bob generates a random $d_B \in \{2, \ldots, m-1\}$ and computes $e_B := g^{d_B}$.
Bob gets the certificate $Cert_B$ on e_B from the certifying authority.
Bob sends $(e_B, Cert_B)$ to Carol.
Carol transmits $((e_B)^d, Cert_B)$ to Alice.
Alice computes $s = ((e_B)^d)^{d_A} = g^{d d_A d_B}$.
Bob computes $s = (e_C)^{d_B} = ((e_A)^d)^{d_B} = g^{d d_A d_B}$.

the public key to an entity (who can prove the knowledge of the corresponding private key) and so the above attack cannot be easily mounted. Nonetheless, the need for stronger authenticated key-exchange protocols is highlighted by the attack.

The Menezes–Qu–Vanstone key-exchange protocol

The *Menezes–Qu–Vanstone* (*MQV*) *key-exchange protocol* is an improved extension of the basic DH scheme, that incorporates public-key authentication. Though the achievement of the desired security goals by the MQV protocol does not seem to be provable, heuristic arguments suggest the effectiveness of the protocol against active adversaries.

Once again, let Alice and Bob be the two parties who plan to agree on a secret element $s \in G$, where the domain parameters G, g, n and m are chosen as in the basic DH scheme. In the MQV scheme, each entity uses two key pairs, one of which ((E_A, D_A) for Alice and (E_B, D_B) for Bob) is called the *static* or the *long-term* key pair, whereas the other ((e_A, d_A) for Alice and (e_B, d_B) for Bob) is called the *ephemeral* or the *short-term* key pair. The static key is bound to an entity for a certain period of time and is used in every invocation of the MQV protocol during that period. On the other hand, each entity generates and uses a new ephemeral key pair during each invocation of the protocol. The static key of an entity is assumed to be authentic, say, certified by a trusted authority. The ephemeral key, on the other hand, is validated using the static private key.

Assume that there is a (publicly known) function $\iota : G \to \mathbb{N}_0$. Let $l := \lfloor \lg m \rfloor + 1$ denote the bit length of $m = \operatorname{ord} g$. For $a \in G$, let $\bar{a}$ denote the integer $2^{\lceil l/2 \rceil} + \big(\iota(a) \operatorname{rem} 2^{\lceil l/2 \rceil}\big)$. The bit size of $\bar{a}$ is about half of that of m. In particular, $\bar{a} \not\equiv 0 \pmod{m}$ for all $a \in G$.

In the MQV protocol, Alice and Bob each computes the shared secret $s = g^{\sigma_A \sigma_B}$, where $\sigma_A := d_A + \bar{e}_A D_A$ and $\sigma_B := d_B + \bar{e}_B D_B$. Here the exponents σ_A and σ_B bear the implicit signatures of Alice and Bob, impressed by their respective static private keys. Alice can compute $s = (g^{\sigma_B})^{\sigma_A} = (e_B (E_B)^{\bar{e}_B})^{\sigma_A}$, since she knows the static

public key E_B and the ephemeral public key e_B of Bob. Similarly, Bob can compute $s = (e_A(E_A)^{\bar{e}_A})^{\sigma_B}$ from a knowledge of the public keys E_A and e_A of Alice. We summarize the steps in Algorithm 5.31.

Algorithm 5.31 **MQV key exchange**

Input: G, g, n and m as defined above.
Output: A secret element $s \in G$ to be shared by Alice and Bob.
Steps:
Alice obtains Bob's static public key E_B.
Bob obtains Alice's static public key E_A.
Alice generates a random integer d_A, $2 \leqslant d_A \leqslant m-1$, and computes $e_A := g^{d_A}$.
Alice sends e_A to Bob.
Bob generates a random integer d_B, $2 \leqslant d_B \leqslant m-1$, and computes $e_B := g^{d_B}$.
Bob sends e_B to Alice.
Alice computes $\sigma_A := d_A + \bar{e}_A D_A \pmod m$.
Alice computes $s := (e_B E_B^{\bar{e}_B})^{\sigma_A}$.
Bob computes $\sigma_B := d_B + \bar{e}_B D_B \pmod m$.
Bob computes $s := (e_A E_A^{\bar{e}_A})^{\sigma_B}$.
if $(s = 1)$ { Return "failure". }

Each participating entity using the MQV protocol performs three exponentiations in G. Alice computes g^{d_A}, $E_B^{\bar{e}_B}$ and $(e_B E_B^{\bar{e}_B})^{\sigma_A}$, of which the first and the last ones have exponents $\mathrm{O}(m)$. On the other hand, $\bar{e}_B$ is $\mathrm{O}(\sqrt{m})$, so that the middle exponentiation is about twice as fast as a full exponentiation. This performance benefit justifies the use of $\bar{e}_A$ and $\bar{e}_B$ instead of e_A and e_B themselves. It appears that using these half-sized exponents does not affect security. Also note that $\bar{e}_A \not\equiv 0 \pmod m$, which implies a non-zero contribution of the static key D_A in the expression σ_A. Similarly for σ_B.

In order to guard against small-subgroup attacks, the MQV algorithm can incorporate the cofactor $k := n/m$, that is, assuming $\gcd(k, m) = 1$, the shared secret would now be $g^{\sigma_A \sigma_B}$ or $g^{k\sigma_A\sigma_B}$, depending on whether compatibility with the original MQV method is desired or not.

The MQV algorithm can be used in a situation when only one party, say, Alice, is capable of initiating a transmission to the other party (Bob). In that case, Bob's static key pair is used also as his ephemeral key pair, that is, the secret element shared between Alice and Bob is $g^{(d_A + \bar{e}_A D_A)(D_B + \bar{E}_B D_B)}$.

See Raymond and Stiglic [250] to know more about the security issues for the DH key agreement protocol and its variants.

EXERCISE SET 5.3

5.12 Let G be a multiplicative Abelian group of order n and with identity 1, H the subgroup of G generated by an element $g \in G$ of order n, $k := n/m$ and $\gcd(k, m) = 1$. Further let a be a non-identity element of G.
(a) Prove that if $a^k = 1$, then $a \notin H$. (The converse of this statement is not true in general, even when G is cyclic. However, if a is an element of small order dividing k, we obviously have $a^k = 1$.)

(b) Explain how the modified Diffie–Hellman protocol (Algorithm 5.29) prevents an active attack by Bob described in connection with small-subgroup attacks.

5.13 Write the MQV key-exchange protocol with cofactor exponentiation.

5.14 Provide the details of the Diffie–Hellman key-exchange algorithm based on the XTR representation (Section 5.2.7).

5.4 Digital Signatures

Suppose an entity (Alice) is required to be bound to some electronic data (like messages or documents or keys). This binding is achieved by Alice *digitally signing* the data in such a way that no party other than Alice would be able to generate the signature. The signature should also be such that any entity can easily verify that it was Alice who generated the signature. *Digital signatures* can be realized using public-key techniques. The entity (Alice) generating a digital signature is called the *signer*, whereas anybody who wants to verify a signature is called a *verifier*.

We have seen in Section 5.2 how the encryption and decryption transforms f_e, f_d achieve confidentiality of sensitive data. If the set of all possible plaintext messages is the same as the set of all ciphertext messages and if f_e and f_d are bijective maps on that set, then the sequence of encryption and decryption can be reversed in order to realize a digital signature scheme. In order to sign m, Alice uses her private key d and the transform f_d to generate $s = f_d(m, d)$. Any party who knows the corresponding public key e can recover m as $m = f_e(s, e)$. This is broadly how a signature scheme works. Depending on how the representative m is generated from the message M that Alice wants to sign, signature schemes can be classified in two categories.

Signature scheme with message recovery In this case, one takes $m = M$. Verification involves getting back the message M. If M is assumed to be (the encoded version of) some human-readable text, then the recovered $M = f_e(s, e)$ will also be human-readable. If s is forged, that is, if a private key $d' \neq d$ has been used to generate $s' = f_d(m, d')$, then verification using Alice's public key yields $m' = f_e(s', e)$, and typically $m' \neq m$, since d' and e are not matching keys. The resulting message m' will, in general, make little or no sense to a human reader. If m is not a human-readable text, one adds some redundancy to it before signing. A forged signature yields m' during verification, which, with high probability, is expected not to have this redundancy.

Attractive as it looks, it is not suitable if M is a long message. In that case, it is customary to break M into smaller pieces and sign each piece separately. Since public-key operations are slow, signature generation (and also verification) will be time-consuming, if there are too many pieces to sign (and verify). This difficulty is overcome using the second scheme described now.

Signature scheme with appendix In this scheme, a *short* representative $m = H(M)$ of M is first computed.[2] The function H is usually chosen to be a *hash function*, that

[2] If M is already a short message, one may go for taking $m = M$. In order to promote uniform treatment, we assume that the function H is *always* applied for the generation of m. Use of H is also desirable from the point of security considerations (Exercise 5.15).

is, one which converts bit strings of arbitrary length to bit strings of a fixed length. H is assumed to be a public knowledge, that is, anybody who knows M can compute m. We also assume that $H(M)$ can be computed fast for messages M of practical sizes. Alice uses the decryption transform on m to generate $s = f_d(m, d)$. The signature now becomes the pair (M, s). A verifier obtains Alice's public key e and checks if $H(M) = f_e(s, e)$. The signature is taken to be valid if and only if equality holds. If a forger uses a private key $d' \neq d$, she generates a signature (M, s'), $s' = f_d(m, d')$, on M and a verifier expects with high probability the inequality $H(M) \neq f_e(s', e)$.

A kind of forgery is possible on signature schemes with appendix. Assume that Alice creates a valid signature (M, s), $s = f_d(H(M), d)$, on a message M. The function H is certainly not injective, since its input space is much bigger (infinite) than its output space (finite). Suppose that Carol finds a message $M' \neq M$ with $H(M') = H(M)$. In that case, the pair (M', s) is a valid signature of Alice on the message M', though it is not Alice who has generated it. (Indeed it has been generated without the knowledge of the private key d of Alice.) In order to foil such attacks, the function H should have second pre-image resistance. The first pre-image resistance and collision resistance properties of a hash functions also turn out to be important in the context of digital signatures. See Sections 1.2.6 and A.4 to know about hash functions.

We now describe some specific algorithms for (generating and verifying) digital signatures. Key pairs used for these algorithms are usually identical to those used for encryption algorithms of Section 5.2 and, therefore, we refrain from a duplicate description of the key-generation procedures. We focus our discussion only on signature schemes with appendix.

5.4.1 The RSA Digital Signature Algorithm

As in the RSA encryption scheme of Section 5.2.1, each entity generates an RSA modulus $n = pq$, which is the product of two distinct large primes p and q. A key pair consists of an encryption exponent e (the public key) and a decryption exponent d (the private key) satisfying $ed \equiv 1 \pmod{\phi(n)}$.

RSA signature generation involves a modular exponentiation in the ring $\mathbb{Z}_n$.

Algorithm 5.32 **RSA signature generation**

Input: A message M to be signed and the signer's private key (n, d).
Output: The signature (M, s) on M.
Steps:
$m := H(M)$. /* $m \in \mathbb{Z}_n$ is the short representative of M */
$s := m^d \pmod{n}$.

Signature generation can be speeded up if the parameters p, q, $d_1 := d \operatorname{rem} (p-1)$, $d_2 := d \operatorname{rem} (q-1)$ and $h := q^{-1} \pmod{p}$ are stored (secretly) in the private key. Now, one can use Algorithm 5.4 for signature generation.

The verification routine also involves a modular exponentiation in $\mathbb{Z}_n$.

Algorithm 5.33 **RSA signature verification**

Input: A signature (M, s) and the signer's public key (n, e).
Output: Verification status of the signature.
Steps:
$m := H(M)$. /* $m \in \mathbb{Z}_n$ is the short representative of M */
$\tilde{m} := s^e \pmod{n}$.
if $(m = \tilde{m})$ { Return "Signature verified". }
else { Return "Signature not verified". }

Small values of e speed up RSA signature verification and are not known to make the scheme suffer from some special attacks. So the values of e like 3, 257 and 65,537 are quite recommended.

5.4.2 The Rabin Digital Signature Algorithm

As in the Rabin encryption algorithm, we choose two distinct large primes p and q of nearly equal sizes and take $n = pq$. The public key is n, whereas the private key is the pair (p, q). The Rabin signature scheme is based on the intractability of computing square roots modulo n in absence of the knowledge of the prime factors p and q of n.

Rabin signature generation involves finding a quadratic residue m modulo n as a representative of the message M and computing a square root of m modulo n.

Algorithm 5.34 **Rabin signature generation**

Input: A message M to be signed and the signer's private key (p, q).
Output: The signature (M, s) on M.
Steps:
$m := H(M)$. /* $m \in \mathbb{Z}_n$ is assumed to be a quadratic residue modulo n */
Compute a square root s_1 of m modulo p. /* Algorithm 3.17 */
Compute a square root s_2 of m modulo q. /* Algorithm 3.17 */
Compute $s \in \mathbb{Z}_n$ satisfying $s \equiv s_1 \pmod{p}$ and $s \equiv s_2 \pmod{q}$. /* CRT */

Verification (Algorithm 5.35) involves a square operation in $\mathbb{Z}_n$.

Algorithm 5.35 **Rabin signature verification**

Input: A signature (M, s) and the signer's public key n.
Output: Verification status of the signature.
Steps:
$m := H(M)$. /* $m \in \mathbb{Z}_n$ is a quadratic residue modulo n */
$\tilde{m} := s^2 \pmod{n}$.
if $(m = \tilde{m})$ { Return "Signature verified". }
else { Return "Signature not verified". }

5.4.3 The ElGamal Digital Signature Algorithm

The ElGamal signature algorithm is based on the intractability of computing discrete logarithms in certain groups G. For a general description, we consider an arbitrary (finite Abelian multiplicative) group G of order n. We assume that G is *cyclic* and that a generator g of G is provided. A key pair is obtained by selecting a random integer (the private key) d, $2 \leqslant d \leqslant n-1$, and then computing g^d (the public key). The hash function H is assumed to convert arbitrary bit strings to elements of $\mathbb{Z}_n$. We further assume that the elements of G can be identified as bit strings (on which the hash function H can be directly applied). G (together with its representation), g and n are considered to be public knowledge and are not input to the signature generation and verification routines.

ElGamal signatures are generated as in Algorithm 5.36. The appendix consists of a pair $(s,t) \in G \times \mathbb{Z}_n$.

Algorithm 5.36 **ElGamal signature generation**

Input: A message M to be signed and the signer's private key d.
Output: The signature (M,s,t) on M.
Steps:
Generate a random session key d', $2 \leqslant d' \leqslant n-1$.
$s := g^{d'}$.
$t := d'^{-1}(H(M) - dH(s)) \pmod{n}$.

The costliest step in the ElGamal signature generation algorithm is the exponentiation $g^{d'}$. Here, G is assumed to be cyclic and the exponent d' to be $\mathrm{O}(n)$. We will shortly see modifications of the ElGamal scheme in which the exponent can be chosen to be much smaller, namely $\mathrm{O}(r)$, where r is a suitably large (prime) divisor of n.

In order to forge a signature, Carol can generate a random session key $(d', g^{d'})$ and obtain s. For the computation of t, she requires the private key d of the signer. Conversely, if t (and d') are available to Carol, she can easily compute the private key d. Thus, forging an ElGamal signature is equivalent to solving the DLP in G.

Each invocation of the ElGamal signature generation algorithm *must* use a new session key $(d', g^{d'})$. If the same session key $(d', g^{d'})$ is used to generate the signatures (M_1, s_1, t_1) and (M_2, s_2, t_2) on two different messages M_1 and M_2, then we have $(t_1 - t_2)d' \equiv H(M_1) - H(M_2) \pmod{n}$, whence d' can be computed, provided that $\gcd(t_1 - t_2, n) = 1$. If d' is known, the private key d can be easily computed (see Exercise 5.6 for a similar situation).

ElGamal signature verification is described in Algorithm 5.37. This is based on the observation that for a (valid) ElGamal signature (M,s,t) on a message M we have $g^{H(M)} = g^{dH(s)+d't} = (g^d)^{H(s)} s^t \in G$. This verification calls for three exponentiations in G to full-size exponents. Working in a suitable (cyclic) subgroup of G makes the algorithm more efficient.

Algorithm 5.37 **ElGamal signature verification**

Input: A signature (M, s, t) and the signer's public key g^d.
Output: Verification status of the signature.
Steps:
$a_1 := g^{H(M)}$.
$a_2 := (g^d)^{H(s)} s^t$.
if $(a_1 = a_2)$ { Return "Signature verified". }
else { Return "Signature not verified". }

ElGamal signatures use a congruence of the form $A \equiv dB + d'C \pmod{n}$, and verification is done by checking the equality $g^A = (g^d)^B s^C$. Our choice for A, B and C was $A = H(M)$, $B = H(s)$ and $C = t$. Indeed, any permutation of $H(M)$, $H(s)$ and t are acceptable as A, B, C. These give rise to several variants of the ElGamal scheme. It is also allowed to take as A, B, C any permutation of $H(M)H(s), t, 1$ or $H(M)H(s), H(M)t, 1$ or $H(M)H(s), H(s)t, 1$ or $H(M)t, H(s)t, 1$. Permutations of $H(M)H(t), H(s), 1$ or $H(M), H(s)t, 1$, on the other hand, are known to have security bugs. For any allowed combination of A, B, C, the choices $\pm A, \pm B, \pm C$ are also valid. For some other variants, see Horster et al. [132].

5.4.4 The Schnorr Digital Signature Algorithm

The *Schnorr signature scheme* is a modification of the ElGamal scheme and is faster than the ElGamal scheme, since it works in a subgroup of G generated by g of *small* order. We assume that $r := \operatorname{ord} g$ is a prime (though it suffices to have $\operatorname{ord} g$ possessing a suitably large prime divisor). We suppose further that the elements of G are represented as bit strings and that we have a hash function H that maps bit strings to elements of $\mathbb{Z}_r^*$. A key pair now consists of an integer d (the private key), $2 \leqslant d \leqslant r - 1$, and the element g^d (the public key).

Schnorr signature generation is described in Algorithm 5.38.

Algorithm 5.38 **Schnorr signature generation**

Input: A message M to be signed and the signer's private key d.
Output: The signature (M, s, t) on M.
Steps:
Generate a random session key pair $(d', g^{d'})$, $2 \leqslant d' \leqslant r - 1$.
$s := H(M||g^{d'})$. /* Here || denotes string concatenation */
$t := d' - ds \pmod{r}$.

Similar to the ElGamal scheme, the most time-consuming step in this routine is the computation of the session public key $g^{d'}$. But now $d' < r$ and, therefore, Algorithm 5.38 runs faster than Algorithm 5.36. One can easily check that forging a signature of Alice is computationally equivalent to determining Alice's private key d from her public key g^d. The importance of using a new session key pair in each run of Algorithm 5.38 is exactly the same as in the case of ElGamal signatures.

The verification of Schnorr signatures (Algorithm 5.39) is based upon the fact that $g^t = g^{d'}(g^d)^{-s}$. Thus, the knowledge of g, s, t and g^d allows one to compute $g^{d'}$ and subsequently $H(M||g^{d'})$. The algorithm involves two exponentiations with both the exponents (t and s) being $\leqslant r$. Thus, signature verification is also faster in the Schnorr scheme than in the ElGamal scheme.

Algorithm 5.39 **Schnorr signature verification**

Input: A signature (M, s, t) and the signer's public key g^d.
Output: Verification status of the signature.
Steps:
$u := g^t(g^d)^s$.
$\tilde{s} = H(M||u)$.
if $(\tilde{s} = s)$ { Return "Signature verified". }
else { Return "Signature not verified". }

5.4.5 The Nyberg–Rueppel Digital Signature Algorithm

The *Nyberg–Rueppel* (*NR*) *signature algorithm* is another adaptation of the ElGamal signature scheme and is based on the intractability of solving the DLP in a group G. We assume that $\operatorname{ord} G = n$ has a large prime divisor r and that an element $g \in G$ of order r is available. Here, a key pair is of the form (d, g^d), where the private key d is an integer between 2 and $r - 1$ (both inclusive) and where the public key g^d is an element of $\langle g \rangle$. The hash function H converts bit strings to elements of $\mathbb{Z}_r$. We also assume the existence of a (publicly known) function $F : G \to \mathbb{Z}_r$.

NR signature generation can be performed as in Algorithm 5.40.

Algorithm 5.40 **Nyberg–Rueppel signature generation**

Input: A message M to be signed and the signer's private key d.
Output: The signature (M, s, t) on M.
Steps:
Generate a random session key pair $(d', g^{d'})$, $2 \leqslant d' \leqslant r - 1$.
$s := H(M) + F(g^{d'}) \pmod r$.
$t := d' - ds \pmod r$.

The only difference between NR signature generation and Schnorr signature generation is the way how s is computed. Therefore, whatever we remarked in connection with the security and the efficiency of the Schnorr scheme applies equally well to the NR scheme. Signature verification is also very analogous, as Algorithm 5.41 explains.

Algorithm 5.41 **Nyberg–Rueppel signature verification**

Input: A signature (M, s, t) and the signer's public key g^d.
Output: Verification status of the signature.
Steps:
$u := g^t(g^d)^s$.
$\tilde{s} := H(M) + F(u) \pmod r$.
if $(\tilde{s} = s)$ { Return "Signature verified". }
else { Return "Signature not verified". }

5.4.6 The Digital Signature Algorithm (DSA)

The *digital signature algorithm* (*DSA*) has been proposed as a standard by the US National Institute of Standards and Technology (NIST) and later accepted as a Federal Information Processing Standard (FIPS) by the US government. This standard is also known as the *digital signature standard* (*DSS*). See the NIST document [220] for a complete description of this standard.

Algorithm 5.42 **Generation of DSA primes**

Input: An integer λ, $0 \leqslant \lambda \leqslant 8$.
Output: A prime p of bit length $l := 512 + 64\lambda$ such that $p - 1$ has a prime divisor r of length 160 bits.
Steps:
Let $l - 1 = 160n + b$, $0 \leqslant b < 160$. /* $n = (l-1) \text{ quot } 160$, $b = (l-1) \text{ rem } 160$. */
while (1) {
  do {
    Choose a random seed σ which is a bit string of length $k \geqslant 160$.
    Compute the bit string $u := H(\sigma) \oplus H((\sigma + 1) \text{ rem } 2^k)$.
    $r := u \text{ OR } 2^{159} \text{ OR } 1$. /* Set the most and least significant bits of u */
  } while (r is not a prime).
  $i := 0$, $f := 2$.
  while $(i < 4096)$ {
    for $j = 0, 1, \ldots, n$ { $v_j := H((\sigma + f + j) \text{ rem } 2^k)$. }
    $v := v_0 + v_1 2^{160} + \cdots + v_{n-1} 2^{160(n-1)} + (v_n \text{ rem } 2^b) 2^{160n} + 2^{l-1}$.
      /* v is an integer of bit length exactly l */
    $p := v - (v \text{ rem } 2r) + 1$. /* $p - 1$ is a multiple of $2r$ */
    if (p is prime) { Return (p, r). }
    $i{+}{+}$, $f := f + n + 1$.
  }
}

DSA is based on the intractability of the DLP in the finite field $\mathbb{F}_p$, where p is a prime of bit length $512 + 64\lambda$ with $0 \leqslant \lambda \leqslant 8$. The cardinality $p - 1$ of $\mathbb{F}_p^*$ is required to have a prime divisor r of length (exactly) 160 bits. The NIST document [220] specifies

a standard method for obtaining such a field $\mathbb{F}_p$, which we describe in Algorithm 5.42. We denote by H the SHA-1 hash function that converts bit strings of arbitrary length to bit strings of length 160. We will identify (often without explicit mention) the bit string $a_1a_2 \ldots a_k$ of length k with the integer $a_1 2^{k-1} + a_2 2^{k-2} + \cdots + a_{k-1}2 + a_k$.

The DSA prime generation procedure (Algorithm 5.42) starts by selecting the prime divisor r and then tries to find a prime p such that $r|(p-1)$. The outputs of H are utilized as pseudorandomly generated bit strings of length 160.

Once the DSA parameters p and r are available, an element $g \in \mathbb{F}_p^*$ of multiplicative order r can be computed by Algorithm 3.26. Henceforth we assume that p, r and g are public knowledge and need not be supplied as inputs to the signature generation and verification routines. A DSA key pair consists of an integer (the private key) d, $2 \leqslant d \leqslant r-1$, and the element g^d (the public key) of $\mathbb{F}_p^*$.

The DSA signature-generation procedure is given as Algorithm 5.43. One may additionally include a check whether $s = 0$ or $t = 0$, and, if so, one should repeat signature generation with another session key. But this, being an extremely rare phenomenon, can be ignored for all practical purposes. Both s and t are elements of $\mathbb{Z}_r$ and hence are represented as integers between 0 and $r-1$.

Algorithm 5.43 **DSA signature generation**

Input: A message M to be signed and the signer's private key d.
Output: The signature (M, s, t) on M.
Steps:
Generate a random session key d', $2 \leqslant d' \leqslant r-1$.
$s := \left(g^{d'} \pmod p\right) \pmod r$.
$t := d'^{-1}(H(M) + ds) \pmod r$.

DSA signature verification is described in Algorithm 5.44. For a valid signature (M, s, t) on a message M, the algorithm computes $w \equiv d'(H(M) + ds)^{-1} \pmod r$, $w_1 \equiv H(M)w \pmod r$ and $w_2 \equiv sw \pmod r$. Therefore, $g^{w_1}(g^d)^{w_2} \equiv g^{w_1+dw_2} \equiv g^{w(H(M)+ds)} \equiv g^{d'(H(M)+ds)^{-1}(H(M)+ds)} \equiv g^{d'} \pmod p$. Reduction modulo r now gives $\tilde{s} = s$.

Algorithm 5.44 **DSA signature verification**

Input: A signature (M, s, t) and the signer's public key g^d.
Output: Verification status of the signature.
Steps:
if ($s \notin \{1, \ldots, r-1\}$ or $t \notin \{1, \ldots, r-1\}$) { Return "Signature not verified". }
$w := t^{-1} \pmod r$.
$w_1 := H(M)w \pmod r$.
$w_2 := sw \pmod r$.
$\tilde{s} := \left(g^{w_1}(g^d)^{w_2} \pmod p\right) \pmod r$.
if ($\tilde{s} = s$) { Return "Signature verified". }
else { Return "Signature not verified". }

DSA signature generation performs a single exponentiation and DSA verification does two exponentiations modulo p. All the exponents are positive and $\leqslant r$. Thus, DSA is essentially as fast as the Schnorr scheme or the NR scheme.

* 5.4.7 The Elliptic Curve Digital Signature Algorithm (ECDSA)

The ECDSA is the elliptic curve analog of the DSA. Algorithm 5.45 describes the generation of the domain parameters necessary to set up an ECDSA system. One first selects a suitable finite field $\mathbb{F}_q$ and takes a random elliptic curve E over $\mathbb{F}_q$. E must be such that the cardinality n of the group $E(\mathbb{F}_q)$ has a suitably large prime divisor r. One generates a random point $P \in E(\mathbb{F}_q)$ of order r and works in the subgroup $\langle P \rangle$ of $E(\mathbb{F}_q)$ generated by P. It is assumed that q is either a prime p or a power 2^m of 2.

Algorithm 5.45 **Generation of ECDSA parameters**

Input: A finite field $\mathbb{F}_q$, where q is a prime p or a power 2^m of 2.
Output: A set of parameters E, n, r, P for the ECDSA.
Steps:

```
while (1) {
  Choose a, b ∈ F_q randomly.
  Consider the curve E : { Y^2 = X^3 + aX + b        if q = p,
                         { Y^2 + XY = X^3 + aX^2 + b  if q = 2^m.
  Compute n := ord E(F_q).
  if (n has a prime divisor r > max(2^160, 4√q)) {
    if (n ∤ (q^k − 1) for k = 1, ..., 20) and (n ≠ q) {
      do {
        Select P' ∈ E(F_q) randomly.
        P := (n/r)P'.
      } while (P = O).
    }
  }
}
```

The order $n = \operatorname{ord} E(\mathbb{F}_q)$ can be computed using the SEA algorithm (for $q = p$) or the Satoh–FGH algorithm (for $q = 2^m$) described in Section 3.6. The integer n should be factored to check if it has a prime divisor $r > \max(2^{160}, 4\sqrt{q})$. The condition $n \nmid (q^k - 1)$ for small values of k is necessary to avoid the MOV attack, whereas the condition $n \neq q$ ensures that the SmartASS attack cannot be mounted. $E(\mathbb{F}_q)$ is not necessarily a cyclic group. But, r being a prime, a point $P = (n/r)P' \in E(\mathbb{F}_q) \setminus \mathcal{O}$ must be one of order r.

An ECDSA key pair consists of a private key d (an integer in the range $2 \leqslant d \leqslant r - 1$) and the corresponding public key $dP \in E(\mathbb{F}_q)$. H denotes the hash function SHA-1 that converts bit strings of arbitrary length to bit strings of length 160. As discussed in connection with DSA, we identify bit strings with integers. We also make an association of elements of $\mathbb{F}_q$ with integers in the set $\{0, 1, \ldots, q - 1\}$. ECDSA signatures can be generated as in Algorithm 5.46. It is necessary to check the conditions

$s \neq 0$ and $t \neq 0$. If these conditions are not both satisfied, one should re-run the procedure with a new session key pair.

Algorithm 5.46 **ECDSA signature generation**

Input: A message M to be signed and the signer's private key d.
Output: The signature (M, s, t) on M.
Steps:
Generate a random session key pair $(d', d'P)$, $2 \leqslant d' \leqslant r-1$.
/* Let us denote $d'P = (h, k) \in E(\mathbb{F}_q)$ */
$s := h \pmod r$.
$t := d'^{-1}(H(M) + ds) \pmod r$.

ECDSA signature verification is explained in Algorithm 5.47. The correctness of this algorithm can be proved like that of Algorithm 5.44.

Algorithm 5.47 **ECDSA signature verification**

Input: A signature (M, s, t) and the signer's public key dP.
Output: Verification status of the signature.
Steps:
if ($s \notin \{1, \ldots, r-1\}$ or $t \notin \{1, \ldots, r-1\}$) { Return "Signature not verified". }
$w := t^{-1} \pmod r$.
$w_1 := H(M)w \pmod r$.
$w_2 := sw \pmod r$.
$Q := w_1 P + w_2(dP)$.
if ($Q = \mathcal{O}$) { Return "Signature not verified". }
/* Otherwise denote $Q = (\tilde{h}, \tilde{k}) \in E(\mathbb{F}_q)$ */
$\tilde{s} := \tilde{h} \pmod r$.
if ($\tilde{s} = s$) { Return "Signature verified". }
else { Return "Signature not verified". }

* 5.4.8 The XTR Signature Algorithm

As discussed in Section 5.2.7, the XTR family of algorithms is an adaptation of other conventional algorithms over finite fields. XTR achieves a speed-up of about three using a clever way of representing elements in certain finite fields. It is no surprise that the DLP-based signature algorithms, described so far, can be given efficient XTR renderings. We explain here *XTR–DSA*, the XTR version of the digital signature algorithm.

In order to set up an XTR system, we need a prime $p \equiv 2 \pmod 3$. The XTR group G is a subgroup of the multiplicative group $\mathbb{F}_{p^6}^*$ and has a prime order q dividing $p^2 - p + 1$. For compliance with the original version of DSA, one requires q to be of bit length 160. The trace map $G \to \mathbb{F}_{p^2}$ taking $h \mapsto \mathrm{Tr}(h) := \mathrm{Tr}_{\mathbb{F}_{p^6} | \mathbb{F}_{p^2}}(h)$ is used to represent an element $h \in G$ by the element $\mathrm{Tr}(h) \in \mathbb{F}_{p^2}$. Under this representation, arithmetic in G translates to that in $\mathbb{F}_{p^2}$. For example, we have seen how exponentiation in G can be efficiently implemented using $\mathbb{F}_{p^2}$ arithmetic (Algorithm 5.20). The trace

$\mathrm{Tr}(g)$ of a generator g of G should also be made available for setting up the XTR domain parameters. In Section 5.2.7, we have discussed how a random set of XTR parameters $(p, q, \mathrm{Tr}(g))$ can be computed.

An *XTR key* comprises a random integer $d \in \{2, 3, \ldots, q-2\}$ (the private key) and the trace $\mathrm{Tr}(g^d) \in \mathbb{F}_{p^2}$ (the public key). Algorithm 5.20 is used to compute $\mathrm{Tr}(g^d)$ from $\mathrm{Tr}(g)$ and d. This algorithm gives $\mathrm{Tr}(g^{d-1})$ and $\mathrm{Tr}(g^{d+1})$ as by-products. For an implementation of XTR–DSA, we require these two elements of $\mathbb{F}_{p^2}$. So we assume that the public key consists of the three traces $S_d(\mathrm{Tr}(g)) = (\mathrm{Tr}(g^{d-1}), \mathrm{Tr}(g^d), \mathrm{Tr}(g^{d+1})) \in \mathbb{F}_{p^2}^3$. As explained in Lenstra and Verheul [172], the values $\mathrm{Tr}(g^{d-1})$ and $\mathrm{Tr}(g^{d+1})$ can be computed easily from $\mathrm{Tr}(g^d)$ even when d is *unknown*, so it suffices to store only $\mathrm{Tr}(g^d)$ as the public key. But we avoid the details of this computation here and assume that all the three traces are available to the signature verifier.

Algorithm 5.20 provides an efficient way of computing exponentiations in G. For DSA-like signature verification (cf. Algorithm 5.44), one computes products of the form $g^a(g^d)^b$ with d unknown. In the XTR world, this amounts to computing the trace $\mathrm{Tr}(g^a(g^d)^b)$ from the knowledge of a, b, $\mathrm{Tr}(g)$ and $\mathrm{Tr}(g^d)$ (or $S_d(\mathrm{Tr}(g))$) but without the knowledge of d. The XTR exponentiation algorithm is as such not applicable in such a situation. We should, therefore, prescribe a method to compute traces of products in G. Doing that requires some mathematics that we mention now without proofs. See Lenstra and Verheul [170] for the missing details.

Let $e := ab^{-1} \pmod q$. Then, $a + bd \equiv b(e + d) \pmod q$, that is, $\mathrm{Tr}(g^a(g^d)^b) = \mathrm{Tr}(g^{b(e+d)})$, that is, it is sufficient to compute $\mathrm{Tr}(g^{e+d})$ from the knowledge of e, $\mathrm{Tr}(g)$ and $\mathrm{Tr}(g^d)$. We treat the 3-tuple $S_k(\mathrm{Tr}(g))$ as a row vector (over $\mathbb{F}_{p^2}$). For $c \in \mathbb{F}_{p^2}$, let M_c denote the matrix

$$M_c := \begin{pmatrix} c_{-2} & c_{-1} & c \\ c_{-1} & c_0 & c_1 \\ c_0 & c_1 & c_2 \end{pmatrix}. \tag{5.9}$$

We take $c := \mathrm{Tr}(g)$. It can be shown that $\det M_{\mathrm{Tr}(g)} = \left[\mathrm{Tr}(g^{p+1})^p - \mathrm{Tr}(g^{p+1})\right]^2 \in \mathbb{F}_{p^2}^*$, that is, the matrix $M_{\mathrm{Tr}(g)}$ is invertible, and we have:

$$\mathrm{Tr}(g^{e+d}) = S_d(\mathrm{Tr}(g)) \cdot M_{\mathrm{Tr}(g)}^{-1} \cdot S_e(\mathrm{Tr}(g))^{\mathrm{t}}. \tag{5.10}$$

Here the superscript $^{\mathrm{t}}$ denotes the transpose of a matrix. With these observations, one can write the procedure for computing $\mathrm{Tr}(g^a(g^d)^b)$ as in Algorithm 5.48.

Algorithm 5.48 **XTR multiplication**

Input: a, b, $\mathrm{Tr}(g)$ and $S_d(\mathrm{Tr}(g))$ for some unknown d.
Output: $\mathrm{Tr}(g^a(g^d)^b)$.
Steps:
Compute $e := ab^{-1} \pmod q$.
Compute $S_e(\mathrm{Tr}(g))$ using Algorithm 5.20 with $c := \mathrm{Tr}(g)$ and $n := e$.
Use Equation (5.10) to compute $\mathrm{Tr}(g^{e+d})$.
Use Algorithm 5.20 with $c := \mathrm{Tr}(g^{e+d})$ and $n := b$ to compute
$S_b(\mathrm{Tr}(g^{e+d})) = \left(\mathrm{Tr}(g^{(b-1)(e+d)}), \mathrm{Tr}(g^{b(e+d)}), \mathrm{Tr}(g^{(b+1)(e+d)})\right) \in \mathbb{F}_{p^2}^3$.
Return $\mathrm{Tr}(g^{b(e+d)})$.

XTR–DSA signature generation (Algorithm 5.49) is an obvious adaptation of Algorithm 5.43.

Algorithm 5.49 **XTR signature generation**

Input: A message M to be signed and the signer's private key d.
Output: The signature (M, s, t) on M with $s, t \in \mathbb{Z}_q$.
Steps:
do {
  Generate a random $d' \in \{2, \ldots, q-1\}$.
  Compute $\mathrm{Tr}(g^{d'})$. /* Use Algorithm 5.20 with $c := \mathrm{Tr}(g)$ and $n := d'$ */
  Let $\mathrm{Tr}(g^{d'}) = x_1\alpha + x_2\alpha^2$. /* α is defined in Section 5.2.7 to represent $\mathbb{F}_{p^2}$ */
  $s := x_1 + px_2 \pmod q$.
} while ($s \neq 0$).
$t := d'^{-1}(H(M) + ds) \pmod q$. /* Here H is the hash function SHA-1 */

The bulk of the time taken by Algorithm 5.43 goes for the computation of $\mathrm{Tr}(g^{d'})$. Since the trace representation of XTR makes this exponentiation three times as efficient as the corresponding DSA exponentiation, XTR–DSA signature generation runs nearly three times as fast as DSA signature generation.

XTR–DSA signature verification can be easily translated from Algorithm 5.44 and is shown in Algorithm 5.50. The most costly step in the XTR–DSA verification routine is the computation of $\mathrm{Tr}(g^{w_1}(g^d)^{w_2})$. One uses Algorithm 5.48 for this purpose. This algorithm, in turn, invokes the exponentiation Algorithm 5.20 twice. For the original DSA signature verification (Algorithm 5.44), the costliest step is the computation of $g^{w_1}(g^d)^{w_2}$, which involves two exponentiations and a (cheap) multiplication. A careful analysis shows that XTR–DSA signature verification runs nearly 1.75 times faster than DSA verification.

Algorithm 5.50 **XTR signature verification**

Input: XTR–DSA signature (M, s, t) on a message M and the signer's public key $(\mathrm{Tr}(g^{d-1}), \mathrm{Tr}(g^d), \mathrm{Tr}(g^{d+1}))$.
Output: Verification status of the signature.
Steps:
if $s \notin \{1, \ldots, q-1\}$ or $t \notin \{1, \ldots, q-1\}$ { Return "Signature not verified". }
$w := t^{-1} \pmod q$.
$w_1 := H(M)w \pmod q$.
$w_2 := sw \pmod q$.
Compute $\mathrm{Tr}(g^{w_1}(g^d)^{w_2})$. /* Use Algorithm 5.48 */
Write this trace value as $\tilde{x}_1\alpha + \tilde{x}_2\alpha^2$. /* See Section 5.2.7 */
$\tilde{s} := \tilde{x}_1 + p\tilde{x}_2 \pmod q$.
if ($\tilde{s} = s$) { Return "Signature verified". }
else { Return "Signature not verified". }

* 5.4.9 The NTRUSign Algorithm

The *NTRU Signature Scheme* (*NSS*) (Hoffstein et al. [131]) is an adaptation of the NTRU encryption algorithm discussed in Section 5.2.8. Cryptanalytic studies (Gentry et al. [110]) show that the NSS has security flaws. A newer version of the NSS, referred to as *NTRUSign* and resistant to these attacks, has been proposed by Hoffstein et al. [128]. In this section, we provide a brief overview of NTRUSign.

In order to set up the domain parameters for NTRUSign, we start with an $n \in \mathbb{N}$ and consider the ring $R := \mathbb{Z}[X]/\langle X^n - 1\rangle$. Elements of R are polynomials with integer coefficients and of degrees $\leqslant n-1$. The multiplication of R is denoted by $\circledast$, which is essentially the multiplication of two polynomials of $\mathbb{Z}[X]$ followed by setting $X^n = 1$. We also fix a positive integer β to be used as a modulus for the coefficients of the polynomials in R. The subsets $\mathcal{R}_f := \mathcal{R}(\nu_f)$ and $\mathcal{R}_g := \mathcal{R}(\nu_g)$ of R are of importance for the NTRUSign algorithm, where for $\nu \in \mathbb{N}$ one defines $\mathcal{R}(\nu) := \{f \in R \mid \nu \text{ coefficients of } f \text{ are } 1, \text{the rest } 0\}$, and where ν_f and ν_g are suitably chosen parameters. The message space $\mathcal{R}_m$ is assumed to consist of *pairs* of polynomials of R with coefficients reduced modulo β. We further assume that we have at our disposal a hash function H that maps messages (that is, binary strings) to elements of $\mathcal{R}_m$.

Let $a(X) := a_0 + a_1 X + \cdots + a_{n-1}X^{n-1} \in R$. The average of the coefficients of a is denoted by $\mu_a := \frac{1}{n}\sum_{i=0}^{n-1} a_i$. The *centred norm* $||a||$ of a is defined by

$$||a|| := \sum_{i=0}^{n-1}(a_i - \mu_a)^2 = \sum_{i=0}^{n-1} a_i^2 - \frac{1}{n}\left(\sum_{i=0}^{n-1} a_i\right)^2 .$$

For two polynomials $a, b \in R$, one also defines

$$||(a,b)||^2 := ||a||^2 + ||b||^2.$$

The parameters ν_f and ν_g should be so chosen that any polynomial $f \in \mathcal{R}_f$ and any polynomial $g \in \mathcal{R}_g$ have (centred) norms on the order $\mathrm{O}(n)$. An upper bound B on the norms (of pairs of polynomials) should also be predetermined.

Typical values for NTRUSign parameters are

$$(n, \beta, \nu_f, \nu_g, B) = (251, 128, 73, 71, 300)\,.$$

It is estimated that these choices lead to a security level at least as high as in an RSA scheme with a 1024-bit modulus. For very long-term security, one may go for $(n, \beta) = (503, 256)$.

In order to set up a key pair, the signer first chooses two random polynomials $f \in \mathcal{R}_f$ and $g \in \mathcal{R}_g$. The polynomial f should be invertible modulo β and the signer computes $f_\beta \in R$ with the property that $f_\beta \circledast f \equiv 1 \pmod{\beta}$. The public key of the signer is the polynomial $h \equiv f_\beta \circledast g \pmod{\beta}$, whereas the private key is the tuple (f, g, F, G), where F and G are two polynomials in R satisfying

$$f \circledast G - g \circledast F = q \quad \text{and} \quad ||F||, ||G|| = \mathrm{O}(n)\,.$$

Hoffstein et al. [128] present an algorithm to compute F and G with $||F||, ||G|| \approx cn/\sqrt{12}$ from polynomials f and g with $||f||, ||g|| \approx c\sqrt{n}$, where c is a given constant.

Algorithm 5.51 **NTRU signature generation**

Input: A message M to be signed and the signer's private key (f, g, F, G).
Output: The signature (M, s) on M.
Steps:
Compute $(m_1, m_2) := H(M) \in \mathcal{R}_m$.
Compute polynomials $A, B, a, b \in R$ satisfying

$$\begin{aligned} G \circledast m_1 - F \circledast m_2 &= A + \beta B, \\ -g \circledast m_1 + f \circledast m_2 &= a + \beta b, \end{aligned}$$

where a and A have coefficients in the range between $-\beta/2$ and $+\beta/2$.
Compute $s \equiv f \circledast B + F \circledast b \pmod{\beta}$.

NTRUSign signature generation is described in Algorithm 5.51. It is apparent that the NTRUSign algorithm derives its security from the difficulty in computing a vector $\mathbf{v}$ in a certain lattice, *close* to the vector defined by the hashed message (m_1, m_2). For defining the lattice, we first note that a polynomial $u(X) = u_0 + u_1 X + \cdots + u_{n-1}X^{n-1} \in R$ can be identified as a vector $(u_0, u_1, \ldots, u_{n-1})$ of dimension n defined by its coefficients. Similarly, two polynomials $u, v \in R$ define a vector, denoted by (u, v), of dimension $2n$. To the public key h we associate the $2n$-dimensional lattice

$$L_h := \{(u, v) \in R^2 \mid u(X) \circledast h(X) \equiv v(X) \pmod{\beta}\} \subseteq \mathbb{R}^{2n}.$$

It is clear from the definitions that both (f, g) and (F, G) are in L_h.

If $h = (h_0, h_1, \ldots, h_{n-1})$, then for each $i = 0, 1, \ldots, n-1$ we have

$$\begin{aligned} X^i \circledast h(X) &\equiv (h_{n-i}, \ldots, h_{n-1}, h_0, \ldots, h_{n-i-1}) \pmod{\beta} \text{ and} \\ 0 \circledast h(X) &\equiv \beta X^i \pmod{\beta}. \end{aligned}$$

It follows immediately that L_h is generated by the rows of the matrix

$$\begin{pmatrix} 1 & 0 & \cdots & 0 & h_0 & h_1 & \cdots & h_{n-1} \\ 0 & 1 & \cdots & 0 & h_{n-1} & h_0 & \cdots & h_{n-2} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & h_1 & h_2 & \cdots & h_0 \\ 0 & 0 & \cdots & 0 & \beta & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 0 & \beta & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & \beta \end{pmatrix}.$$

Now, consider the signature generation routine (Algorithm 5.51). The hash function H generates from the message M a random $2n$-dimensional vector $\mathbf{m} := (m_1, m_2)$ not necessarily on L_h. We then look at the vector $\mathbf{v} := (s, t)$ defined as:

$$\begin{aligned} s &\equiv f \circledast B + F \circledast b \pmod{\beta}, \text{ and} \\ t &\equiv g \circledast B + G \circledast b \pmod{\beta}. \end{aligned}$$

The lattice L_h has the *rotational invariance property*, namely, if $(u, v) \in L_h$, then $(X^i \circledast u, X^i \circledast v)$ is also in L_h for all $i = 0, 1, \ldots, n-1$. More generally, if $(u, v) \in L_h$, then $(w \circledast u, w \circledast v) \in L_h$ for any polynomial $w \in R$. In particular, since $\mathbf{v} = (s, t) = B \circledast (f, g) + b \circledast (F, G) \pmod{\beta}$ and since $(f, g), (F, G) \in L_h$, it follows that $\mathbf{v} \in L_h$. Of these two polynomials only s is needed for the generation of NTRUSign signatures. The other is needed during signature verification and can be computed easily from s using the formula $t \equiv h \circledast s \pmod{\beta}$, the validity of which is established from the definition of the lattice L_h.

The vector $\mathbf{v} \in L_h$ is close to the message vector $\mathbf{m}$ in the sense that

$$||(m_1 - s, m_2 - t)||^2 = ||m_1 - s||^2 + ||m_2 - t||^2 \approx \frac{c^2 n^3}{72} \left(1 + \frac{12}{n}\right)$$

for the constant c chosen earlier (see Hoffstein et al. [128] for a proof of this relation). The verification routine can, therefore, be designed as in Algorithm 5.52.

Algorithm 5.52 **NTRU signature verification**

Input: A signature (M, s) and the signer's public key h.
Output: Verification status of the signature.
Steps:
Compute $(m_1, m_2) := H(M) \in \mathcal{R}_m$.
Compute $t \equiv h \circledast s \pmod{\beta}$.
if $(||(m_1 - s, m_2 - t)|| \leqslant B)$ { Return "Signature verified". }
else { Return "Signature not verified". }

For the choice $(n, \beta, c) = (251, 128, 0.45)$, we have $||(m_1 - s, m_2 - t)|| \approx 216$. Therefore, choosing the norm bound B slightly larger than this value (say, $B = 300$) allows the verification scheme to work correctly most of the time. The knowledge of the private key (f, g, F, G) allows the legitimate signer to compute the close vector (s, t) easily. On the other hand, for a forger (who is lacking the private information) fast computation of a vector $\mathbf{v}' = (s', t')$ with small norm $||(m_1 - s', m_2 - t')||$ (say $\leqslant 400$ for the above parameter values) seems to be an intractable task. This is precisely why forging an NTRUSign signature is considered infeasible.

An exhaustive search can be mounted for generating a valid signature (s', t') on a message M with $H(M) = (m_1, m_2)$. More precisely, a forger fixes half of the $2n$ coefficients of the polynomials s' and t' and then tries to solve $t' \equiv h \circledast s' \pmod{\beta}$ for the remaining half such that the norm $||(m_1 - s', m_2 - t')||$ is small. It is estimated (see Hoffstein et al. [128] for the details) that the probability that a random guess for the unknown half succeeds is very low ($\leqslant 2^{-178.44}$ for the given parameter values).

Another attack on the NTRUSign scheme is to determine the polynomials f, g from a knowledge of h. Since (f, g) is a short non-zero vector in the lattice L_h, an algorithm that can find such vectors can determine (f, g) (or a rotated version of it). However, for a proper choice of the parameters such an algorithm is deemed infeasible. (Also see the NTRU encryption scheme in Section 5.2.8.)

Similar to the NTRU encryption scheme, the NTRUSign scheme is fast, namely, both signature generation and verification can be carried out in time $O(n^2)$. This is one

of the main reasons why the NTRUSign scheme deserves popularity. Indeed, it may be adopted as an IEEE standard. Unfortunately, however, several attacks on NTRUSign are known. Gentry and Szydlo [111] indicate the possibility of extending the attacks of Gentry et al. [110]. Nguyen [217] proposes a more concrete attack on NTRUSign, that is capable of recovering the private key from only 400 signatures. The future of NTRUSign and its modifications remains uncertain.

5.4.10 Blind Signature Schemes

Suppose that an entity (Alice) referred to as the *sender* or the *user*, wants to get a message M signed by a second entity (Bob) called the *signer*, without revealing M to Bob. This can be achieved as follows. First Alice transforms the message M to $\bar{m} = f(M)$ and sends $\bar{m}$ to Bob. Bob generates the signature $(\bar{m}, \sigma)$ on $\bar{m}$ and sends this pair back to Alice. Finally, Alice applies a second transform g to generate the signature $(M, s) = g(\bar{m}, \sigma)$ of Bob on M. The transform f hides the actual message M from Bob and, thereby, disallows Bob from associating Alice with the signed message (M, s). Such a signature scheme is called a *blind signature scheme*.

Blind signatures are widely used in electronic payment systems in which Alice (a customer) wants the signature of Bob (the bank) on an electronic coin, but does not want the bank to be capable of associating Alice with the coin. In this way, Alice achieves *anonymity* while spending an electronic coin.

In a blind signature scheme, Bob does not know M, but his signature on $\bar{m} = f(M)$ is essential for Alice to reconstruct the signature on M. Furthermore, the blind signature on M should not allow Alice to compute the blind signature on another message M'. More generally, Alice should not be able to generate $l + 1$ (or more) blind signatures with only l (or fewer) interactions with Bob. A forgery of this kind is often called an $(l, l + 1)$ *forgery* or a *one-more forgery* (in case l is bounded above by a polynomial in the security parameter) or a *strong one-more forgery* (in case l is bounded above poly-logarithmically in the security parameter). An $(l, l + 1)$ forgery is mountable on a scheme which is not existentially unforgeable (Exercises 5.15 and 5.19). Usually, existential forgery gives forged signatures on messages over which the forger has no (or little) control (that is, on messages which are likely to be meaningless).

Now, we describe some common blind signature schemes. We provide a brief overview of the algorithms. Detailed analysis of the security of these schemes can be found in the references cited at the end of this chapter.

Chaum's RSA blind signature protocol

Chaum's blind signature protocol is based on the intractability of the RSAP (or the IFP). The signer generates two (distinct) large random primes p and q and computes $n := pq$. He then chooses a random integer e with $\gcd(e, \phi(n)) = 1$ and computes an integer d such that $ed \equiv 1 \pmod{\phi(n)}$. The public key (of the signer) is the pair (n, e), whereas the private key is d. Chaum's protocol works as in Algorithm 5.53.

Algorithm 5.53 **Chaum's RSA blind signature**

Input: A message M generated by Alice.
Output: Bob's blind RSA signature (M, s) on M.
Steps:
Alice hashes the message M to $m := H(M) \in \mathbb{Z}_n$.
Alice chooses a random $\rho \in \mathbb{Z}_n^*$ and computes $\bar{m} := \rho^e m \pmod{n}$.
Alice sends $\bar{m}$ to Bob.
Bob generates the signature $\sigma := \bar{m}^d \pmod{n}$ on $\bar{m}$.
Bob sends σ to Alice.
Alice computes Bob's (blind) signature $s := \rho^{-1}\sigma \pmod{n}$ on M.

Since $\sigma \equiv (\rho^e m)^d \equiv \rho m^d \pmod{n}$, we have $s \equiv \rho^{-1}\sigma \equiv m^d \pmod{n}$, that is, s is indeed the RSA signature of Bob on M. Bob receives $\bar{m}$ and gains no idea about m, since ρ is randomly and secretly chosen by Alice.

The Schnorr blind signature protocol

Let G be a finite multiplicative Abelian group and let $g \in G$ be of order r (a large prime). We assume that computing discrete logarithms in G is an infeasible task. The key pair of the signer is denoted by (d, g^d), where the integer d, $2 \leqslant d \leqslant r-1$, is the private key and g^d the public key. The Schnorr blind signature protocol is described in Algorithm 5.54.

Algorithm 5.54 **Schnorr blind signature**

Input: A message M generated by Alice.
Output: Bob's blind Schnorr signature (M, s, t) on M.
Steps:
Alice asks Bob to initiate a communication.
Bob chooses a random $\bar{d} \in \{2, \ldots, r-1\}$ and computes $\bar{u} := g^{\bar{d}}$.
Bob sends $\bar{u}$ to Alice.
Alice selects $\alpha, \beta \in \mathbb{Z}_r$ randomly.
Alice computes $u := \bar{u} g^{\alpha} (g^d)^{\beta}$.
Alice computes $s := H(M||u) \in \mathbb{Z}_r^*$ and $\bar{s} := s - \beta \pmod{r}$.
Alice sends $\bar{s}$ to Bob.
Bob computes $\bar{t} := \bar{d} - d\bar{s} \pmod{r}$.
Bob sends $\bar{t}$ to Alice.
Alice computes $t := \bar{t} + \alpha \pmod{r}$.

It is easy to check that the output (M, s, t) of Algorithm 5.54 is a valid Schnorr signature of Bob on the message M. The session key d' (Algorithm 5.38) for this signature is $d' \equiv \bar{d} + \alpha + \beta d \pmod{r}$. Since d and $\bar{d}$ are secret knowledges of Bob, Alice must depend on Bob for the computation of $\bar{t}$. The message M is never sent to Bob. Also its hash is masked by β. This is how this protocol achieves blindness.

The Okamoto–Schnorr blind signature protocol

Okamoto's adaptation of the Schnorr scheme is proved to be resistant to an attack by a third entity (Pointcheval and Stern [237]). As in the Schnorr scheme, we fix a (finite multiplicative Abelian) group G (in which it is difficult to compute discrete logarithms). We then choose two elements $g_1, g_2 \in G$ of (large prime) order r. The private key of the signer now comprises a pair (d_1, d_2) of integers in $\{2, \ldots, r-1\}$, whereas the public key y is the group element $y = g_1^{d_1} g_2^{d_2}$. We assume that there is a hash function H whose outputs are in $\mathbb{Z}_r$. We identify elements of G as bit strings. The Okamoto–Schnorr blind signature protocol is explained in Algorithm 5.55.

Algorithm 5.55 **Okamoto–Schnorr blind signature**

Input: A message M generated by Alice.
Output: Bob's blind signature (M, s_1, s_2, s_3) on M.
Steps:
Alice asks Bob to initiate a communication.
Bob chooses random $\bar{d}_1, \bar{d}_2 \in \{2, \ldots, r-1\}$ and computes $\bar{u} := g_1^{\bar{d}_1} g_2^{\bar{d}_2}$.
Bob sends $\bar{u}$ to Alice.
Alice selects $\alpha, \beta, \gamma \in \mathbb{Z}_r$ randomly.
Alice computes $u := \bar{u} g_1^{\alpha} g_2^{\beta} y^{\gamma} = \bar{u} g_1^{\alpha} g_2^{\beta} (g_1^{d_1} g_2^{d_2})^{\gamma}$.
Alice computes $s_1 := H(M||u) \in \mathbb{Z}_r^*$ and $\bar{s}_1 := s_1 - \gamma \pmod r$.
Alice sends $\bar{s}_1$ to Bob.
Bob computes $\bar{s}_2 := \bar{d}_1 - d_1 \bar{s}_1 \pmod r$ and $\bar{s}_3 := \bar{d}_2 - d_2 \bar{s}_1 \pmod r$.
Bob sends $\bar{s}_2$ and $\bar{s}_3$ to Alice.
Alice computes $s_2 := \bar{s}_2 + \alpha \pmod r$ and $s_3 := \bar{s}_3 + \beta \pmod r$.

An Okamoto–Schnorr signature (M, s_1, s_2, s_3) on a message can be verified by checking the equality $s_1 = H(M||u)$, where $u := g_1^{s_2} g_2^{s_3} y^{s_1}$. Each invocation of the protocol uses a session private key $(d_1', d_2') := (\bar{d}_1 + \alpha + d_1\gamma, \bar{d}_2 + \beta + d_2\gamma)$. Alice must depend on Bob for generating s_2 and s_3, because she is unaware of the private values d_1, d_2, $\bar{d}_1$ and $\bar{d}_2$. Alice, in an attempt to forge Bob's blind signature, may start with random $\bar{d}_1$ and $\bar{d}_2$ of her choice. But she still needs the integers d_1 and d_2 in order to complete the protocol. The blindness of Algorithm 5.55 stems from the fact that the message M is never sent to Bob and its hash is masked by γ.

5.4.11 Undeniable Signature Schemes

So far we have seen signature schemes for which any entity with a knowledge of the signer's public key can verify the authenticity of a signature. There are, however, situations where an active participation of the signer is necessary for the verification of a signature. Moreover, during a verification interaction a signer should not be allowed to deny a legitimate signature made by him. A signature meeting these requirements is called an *undeniable signature*.

Undeniable signatures are typically used for messages that are too confidential or private to be given unlimited verification facility. In case of a dispute, an entity should be capable of proving a forged signature to be so and at the same time must accept

the binding to his own valid signatures. So in addition to the signature generation and verification protocols, an undeniable signature scheme comes with a *denial* or *disavowal protocol* to guard against a cheating signer that is unwilling to accept his valid signature either by not taking part in the verification interaction or by responding incorrectly or by claiming a valid signature to be forged.

There are applications where undeniable signatures are useful. For example, a software vendor can use undeniable signatures to prove the authenticity of its products only to its (paying) customers (and not to everybody).

Chaum and van Antwerpen gave a first concrete realization of an undeniable signature scheme [52, 51]. It is based on the intractability of computing discrete logs in the group $\mathbb{F}_p^*$, p a prime. Gennaro et al. [109] later adapted the algorithm to design an RSA-based undeniable signature scheme. We now describe these two schemes. Rigorous studies of these schemes can be found in the original papers. See also [53, 186, 187, 102, 202, 230].

The Chaum–Van Antwerpen undeniable signature scheme

For setting up the domain parameters for Chaum–Van Antwerpen (CvA) signatures, Bob chooses a (large) prime p of the form $p = 2r + 1$, where r is also a prime. (Such a prime p is called a safe prime (Definition 3.5).) Bob finds a random element $g \in \mathbb{Z}_p^*$ of multiplicative order r, selects a random integer $d \in \{2, \ldots, r-1\}$ and computes $y := g^d \pmod p$. Bob publishes (p, g, y) as his public key and keeps the integer d secret as his private key. The value $d^{-1} \pmod r$ is needed during verification and can be precomputed and stored (secretly) along with d. We assume that we have a hash function H that maps messages (that is, bit strings) to elements of the subgroup of order r in $\mathbb{Z}_p^*$. In order to generate a CvA signature on a message M, Bob carries out the steps given in Algorithm 5.56. Verification of Bob's CvA signature by Alice involves the interaction given in Algorithm 5.57.

Algorithm 5.56 **Chaum–Van Antwerpen undeniable signature generation**

Input: The message M to be signed and the signer's private key (p, d).
Output: The signature (M, s) on M.
Steps:
$m := H(M)$.
$s := m^d \pmod p$.

If (M, s) is a valid CvA signature, then

$$v \equiv (s^i y^j)^{d^{-1} \pmod r} \equiv ((m^d)^i (g^d)^j)^{d^{-1} \pmod r} \equiv m^i g^j \equiv v' \pmod p.$$

On the other hand, if $s \not\equiv m^d \pmod p$, Bob can guess the element v' with a probability of only $1/r$, even under the assumption that Bob has unbounded computing resources. This means that unless the signature (M, s) is valid, it is extremely unlikely that Bob can make Alice accept the signature.

The denial protocol for the CvA scheme involves an interaction between the prover Bob and the verifier Alice, as given in Algorithm 5.58. In order to see how this

Algorithm 5.57 **Chaum–Van Antwerpen undeniable signature verification**

Input: A CvA signature (M, s) on a message M.
Output: Verification status of the signature.
Steps:
Alice computes $m := H(M)$.
Alice chooses two secret random integers $i, j \in \{1, \ldots, q-1\}$.
Alice computes $u := s^i y^j \pmod p$.
Alice sends u to Bob.
Bob computes $v := u^{d^{-1} \pmod r} \pmod p$.
Bob sends v to Alice.
Alice computes $v' := m^i g^j \pmod p$.
Alice accepts the signature (M, s) if and only if $v = v'$.

Algorithm 5.58 **Chaum–Van Antwerpen undeniable signature: denial protocol**

Input: A (purported) CvA signature (M, s) of Bob on a message M.
Output: One of the following decisions by Alice:
(i) The signature is valid.
(ii) The signature is forged.
(iii) Bob is trying to deny the signature.
Steps:
Alice computes $m := H(M)$.
Alice chooses two secret random integers $i_1, j_1 \in \{1, \ldots, q-1\}$.
Alice computes $u_1 := s^{i_1} y^{j_1} \pmod p$ and sends u_1 to Bob.
Bob computes $v_1 := u_1^{d^{-1} \pmod r} \pmod p$ and sends v_1 to Alice.
if ($v_1 \equiv m^{i_1} g^{j_1} \pmod p$) {
  Alice accepts the signature (M, s) to be valid and quits the protocol.
}
Alice chooses two other secret random integers $i_2, j_2 \in \{1, \ldots, q-1\}$.
Alice computes $u_2 := s^{i_2} y^{j_2} \pmod p$ and sends u_2 to Bob.
Bob computes $v_2 := u_2^{d^{-1} \pmod r} \pmod p$ and sends v_2 to Alice.
if ($v_2 \equiv m^{i_2} g^{j_2} \pmod p$) {
  Alice concludes the signature (M, s) to be valid and quits the protocol.
}
Alice computes $w_1 := (v_1 g^{-j_1})^{i_2} \pmod p$ and $w_2 := (v_2 g^{-j_2})^{i_1} \pmod p$.
if ($w_1 = w_2$) {
  Alice concludes that the signature is forged.
} else {
  Alice concludes that Bob is trying to deny the signature.
}

denial protocol works, we note that Algorithm 5.58 essentially makes two calls of the verification protocol. First assume that Bob executes the protocol honestly, that is, Bob follows the steps as indicated. If the signature (M, s) is a valid one, the check $v_1 \equiv m^{i_1} g^{j_1} \pmod{p}$ (as well as the check $v_2 \equiv m^{i_2} g^{j_2} \pmod{p}$) should succeed and Alice's decision to accept the signature as valid is justified. On the other hand, if (M, s) is a forged signature, that is, if $s \not\equiv m^d \pmod{p}$, then the probability that each of these checks succeeds is $1/r$ as discussed before. Thus, it is extremely unlikely that a forged signature is accepted as valid by Alice. So Alice eventually computes both w_1 and w_2 equal to $s^{i_1 i_2 d^{-1} \pmod{r}} \pmod{p}$ and accepts the signature to be forged. Finally, suppose that Bob is intending to deny the (purported) signature (M, s). If Bob does not fully take part in the interaction, then his intention becomes clear. Otherwise, he sends v_1 and/or v_2 not computed according to the formulas specified. In that case, Bob succeeds in making Alice compute $w_1 = w_2$ with a probability of only $1/r$. Thus, it is extremely unlikely that Bob executing this protocol dishonestly can successfully disavow a valid signature.

RSA-based undeniable signature scheme

Gennaro, Krawczyk and Rabin's undeniable signature scheme (the GKR scheme) is based on the (intractability of the) RSA problem.

A GKR key pair differs from a usual RSA key pair. The signer chooses two (large) random primes p and q such that both $p' := (p-1)/2$ and $q' := (q-1)/2$ are also prime, and sets $n := pq$. Two integers e and d satisfying $ed \equiv 1 \pmod{\phi(n)}$ are then selected. Finally, one requires a $g \in \mathbb{Z}_n^*$, $g \neq 1$, and $y \equiv g^d \pmod{n}$. The public key of the signer is the tuple (n, g, y), whereas the private key is the pair (e, d). It can be shown that g need not be a random element of $\mathbb{Z}_n^*$. Choosing a (fixed) small value of g (for example, $g = 2$) does not affect the security of the GKR protocol, but makes certain operations (computing powers of g) efficient.

Algorithm 5.59 **GKR RSA undeniable signature generation**

Input: The message M to be signed and the signer's private key (e, d).
Output: The signature (M, s) on M.
Steps:
$m := H(M)$. /* Hash the message M to an element m of $\mathbb{Z}_n$ */
$s := m^d \pmod{n}$.

GKR signature generation (Algorithm 5.59) is the same as in RSA. The verification protocol described in Algorithm 5.60 accepts, in addition to a valid GKR signature (M, s), the signatures $(M, \alpha s)$, where $\alpha \in \mathbb{Z}_n^*$ has multiplicative order 1 or 2 (there are four such values of α). In view of this, we define the subset

$$\operatorname{Sig} M := \{\alpha H(M)^d \pmod{n} \mid \alpha \in \mathbb{Z}_n^*, \operatorname{ord} \alpha \leqslant 2\}$$

of $\mathbb{Z}_n$. Any element $s \in \operatorname{Sig} M$ is considered to be a valid signature on M. Since Bob knows p and q, he can easily find out all the elements α of $\mathbb{Z}_n^*$ of order $\leqslant 2$ and can choose to output $(M, \alpha H(M)^d)$ as the GKR signature for any such α. Taking $\alpha = 1$

(as in Algorithm 5.59) is the canonical choice, but during the execution of the denial protocol Bob will not be allowed to disavow other valid choices.

The interaction between the prover Bob and the verifier Alice during GKR signature verification is given in Algorithm 5.60. It is easy to see that if (M, s) is a valid GKR signature, then $v = v'$. On the other hand, if (M, s) is a forged signature, that is, if $s \notin \operatorname{Sig} M$, then the equality $v = v'$ occurs with a probability of $\frac{\mathrm{O}(1)}{\min(p',q')}$, even in the case that the forger has unbounded computational resources.

Algorithm 5.60 **GKR RSA undeniable signature verification**

Input: A GKR signature (M, s) on a message M.
Output: Verification status of the signature.
Steps:
Alice computes $m := H(M)$.
Alice chooses random $i, j \in \{1, \ldots, n\}$.
Alice computes $u := s^{2i} y^j \pmod n$.
Alice sends u to Bob.
Bob computes $v := u^e \pmod n$.
Bob sends v to Alice.
Alice computes $v' := m^{2i} g^j \pmod n$.
Alice accepts the signature (M, s) if and only if $v = v'$.

Algorithm 5.61 **GKR RSA undeniable signature: denial protocol**

Input: A (purported) GKR signature (M, s) of Bob on a message M.
Output: One of the following decisions by Alice:
(i) The signature is forged.
(ii) Bob is trying to deny the signature.
Steps:
Alice computes $m := H(M)$.
Alice chooses random $i \in \{4, 8, 12, \ldots, 4k\}$ and $j \in \{1, \ldots, n\}$.
Alice computes $w_1 := m^i g^j \pmod n$ and $w_2 := s^i y^j \pmod n$.
Alice sends (w_1, w_2) to Bob.
Bob computes $m := H(M)$.
Bob determines $i' \in \{4, 8, 12, \ldots, 4k\}$ such that the following congruence holds:

$$w_1 w_2^{-e} \equiv (m s^{-e})^{i'} \pmod n . \tag{5.11}$$

if (no such i' is found) { /* This may happen, if Alice has cheated */
Bob aborts the protocol.
}
Bob sends i' to Alice.
if $(i = i')$ {
Alice concludes that the signature is forged.
} else {
Alice concludes that Bob is trying to deny the signature.
}

The denial protocol for the GKR scheme is described in Algorithm 5.61. This protocol is executed, after verification by Algorithm 5.60 fails. In that case, Alice wants to ascertain whether the signature is actually invalid or Bob has denied his valid signature by incorrectly executing the verification protocol. A small integer k is predetermined for the denial protocol. The prover needs a running time proportional to k, whereas the probability of a successful denial of a valid signature decreases with k. Taking $k = \mathrm{O}(\lg n)$ gives optimal performance.

In order to see how this protocol prevents Bob from denying a valid signature, first consider the case that (M, s) is a valid GKR signature of Bob. In that case, $w_2^e \equiv (s^i y^j)^e \equiv (\alpha^i m^{di} g^{dj})^e \equiv (m^i g^j)^{de} \equiv m^i g^j \equiv w_1 \pmod{n}$. On the other hand, $s^e \equiv \alpha^e m^{de} \equiv \alpha^e m \pmod{n}$. Therefore, for every $i' \in \{4, 8, \ldots, 4k\}$, one has $w_1 w_2^{-e} \equiv (m s^{-e})^{i'} \equiv 1 \pmod{n}$. Thus, Bob can only guess the secret value of i chosen by Alice and the guess is correct with a probability of $1/k$. On the other hand, if (M, s) is a forged signature, Congruence (5.11) holds only for a single i', that is, for $i' = i$ (Exercise 5.23). Sending i' will then convince Alice that the signature is really forged. In both these cases, Congruence (5.11) holds for at least one i'. Failure to detect such an i' implies that the value(s) of w_1 and/or w_2 have not been correctly sent by Alice. The protocol should then be aborted.

In order to reduce the probability of successful cheating, it is convenient to repeat the protocol few times instead of increasing k. If $k = 1024$, Bob can successfully cheat in eight executions of the denial protocol with a probability of only 2^{-80}.

5.4.12 Signcryption

The conventional way to ensure both authentication and confidentiality of a message is to sign the message first and then encrypt the signed message. Now that we have many signature and encryption algorithms in our bag, there is hardly any problem in achieving both the goals simultaneously. Zheng proposes *signcryption* schemes that *combine* these two operations together. A signcryption scheme is better than a sign-and-encrypt scheme in two aspects. First, the combined primitive takes less running time than the composite primitive comprising signature generation followed by encryption. Second, a signcrypted message is of smaller size than a signed-and-encrypted message. When communication overheads need to be minimized, signcryption proves to be useful.

Before describing the signcryption primitive, let us first review the composite sign-and-encrypt scheme. Let M be the message to be sent. Alice the sender generates the signature appendix s on M using one of the signature schemes described earlier. This step can be described as $s = f_s(M, d_a)$, where d_a is the private key of Alice. Next a symmetric key k is generated by Alice. The message M is encrypted by a symmetric cipher (like DES) under the key k, that is, $C := E(M, k)$. The key k is then encrypted using an asymmetric routine under the public-key e_b of Bob the recipient, that is, $c = f_e(k, e_b)$. The triple (C, c, s) is then transmitted to Bob.

Upon reception of (C, c, s) Bob first retrieves k using his private key d_b, that is, $k = f_d(c, d_b)$. The message M is then recovered by symmetric decryption: $M = D(C, k)$. Finally, the authenticity of M is verified from the signature using the verification operation: $f_v(M, s, e_a)$, where e_a is the public key of Alice. Algorithm 5.62 describes the sign-and-encrypt operation and its inverse.

Algorithm 5.62

Sign-and-encrypt

$s := f_s(M, d_a)$.
Generate a random symmetric key k.
$c := f_e(k, e_b)$.
$C := E(M, k)$.
Send (C, c, s) to the recipient.

Decrypt-and-verify

$k := f_d(c, d_b)$.
$M := D(C, k)$.
Verify the signature: $f_v(M, s, e_a)$.

Zheng's signcryption scheme combines f_s and f_e to a single operation f_{se} and also f_d and f_v to another single operation f_{dv}. Each of these combined operations essentially takes the time of a single public- or private-key operation and hence leads to a performance enhancement by a factor of nearly two. Moreover, the encrypted key c need not be sent with the message, that is, C and s are sufficient for both authentication and confidentiality. This reduces communication overhead.

Table 5.3 Shortened digital signature algorithms

Name	Signature generation	Signature verification
SDSA1	$s := H(g^{d'} \pmod p) \| \| M)$.	$w := (e_a g^s)^t \pmod p$.
	$t := d'(s + d)^{-1} \pmod r$.	Verify if $s = H(w \| \| M)$.
SDSA2	$s := H(g^{d'} \pmod p) \| \| M)$.	$w := (g e_a^s)^t \pmod p$.
	$t := d'(1 + ds)^{-1} \pmod r$.	Verify if $s = H(w \| \| M)$.

Signcryption is based on shortened digital signature schemes. Table 5.3 describes the shortened version of DSA (Section 5.4.6). We use the notations of Algorithms 5.43 and 5.44. Also $||$ denotes concatenation of strings, and H is a hash function (like SHA-1). The shortened schemes have two advantages over the original DSA. First, a DSA signature is of length $2|r|$, whereas an SDSA1 or SDSA2 signature has length $|r| + |H(\cdot)|$. For the current version of the standard, both r and $H(\cdot)$ are of size 160 bits. However, one may use potentially bigger r and in that case the shortened schemes give smaller signatures with equivalent security. Finally, DSA requires computing a modular inverse during verification, whereas SDSA does not. So verification is more efficient in the shortened schemes.

Algorithms 5.63 and 5.64 provide the details of the signcryption algorithm and its inverse called *unsigncryption*. The algorithms use a keyed hash function KH. One may implement $KH(x, \kappa)$ as $H(\kappa||x)$ using an unkeyed hash function H.

Signcryption differs from the shortened scheme in that $e_b^{d'}$ is used instead of $g^{d'}$ for the computation of s. The running time of the signcryption algorithm is dominated by this modular exponentiation. When signature and encryption are used separately, the encryption operation uses one (or more) exponentiations. So signcryption significantly improves upon the sign-and-encrypt scheme of Algorithm 5.62.

Algorithm 5.63 **Signcryption**

Input: Plaintext message M, the sender's private key d_a, the recipient's public key $e_b = g^{d_b} \pmod p$.
Output: The signcrypted message (C, s, t).
Steps:
Select a random $d' \in \{2, 3, \ldots, r-1\}$.
$k := H(e_b^{d'} \pmod p)$. /* Generate keys for both signing and encrypting. */
Write $k := k_1 \,||\, k_2$ with $|k_2|$ equal to the length of a symmetric key.
$s := KH(M||N, k_1)$.
/* Here N is the public key or the public key certificate of the sender. */
$t := \begin{cases} d'(s + d_a)^{-1} \pmod r & \text{if SDSA1 is used,} \\ d'(1 + d_a s)^{-1} \pmod r & \text{if SDSA2 is used.} \end{cases}$
$C := E(M, k_2)$. /* Symmetric encryption */

Algorithm 5.64 **Unsigncryption**

Input: The signcrypted message (C, s, t), the sender's public key $e_a = g^{d_a} \pmod p$ and the recipient's private key d_b.
Output: The plaintext message M and the verification status of the signature.
Steps:
Recover the keys: $k = \begin{cases} (e_a g^s)^{td_b} \pmod p & \text{if SDSA1 is used,} \\ (g e_a^s)^{td_b} \pmod p & \text{if SDSA2 is used.} \end{cases}$
Write $k := k_1 \,||\, k_2$ with $|k_2|$ equal to the length of a symmetric key.
$M := D(C, k_2)$. /* Symmetric decryption */
if ($KH(M||N, k_1) = s$) { Return "Signature verified". }
else { Return "Signature not verified". }

The most time-consuming part of unsigncryption is the computation of two modular exponentiations. DSA verification too has this property. However, an additional decryption in the decrypt-and-verify scheme of Algorithm 5.62 calls for one (or more) exponetiations, making it slower that unsigncryption.

EXERCISE SET 5.4

5.15 **(a)** Show how first pre-image resistance of the hash function H plays an important role for RSA signatures (with appendix) described in Section 5.4.1. More precisely, show that if it is easy to find a pre-image of any hash value, it is easy to generate a valid signature (M, s) from two valid signatures (M_1, s_1) and (M_2, s_2) with $M \notin \{M_1, M_2\}$. This is often referred to as *existential forgery* of a signature. **[H]**
(b) Describe how existential forgery is possible for the Rabin signature scheme. **[H]**
(c) Describe how existential forgery is possible for the ElGamal signature scheme. **[H]**

5.16 Assume that Bob uses the same RSA key pair $((n, e), d)$ for receiving encrypted messages and for signing. Suppose that Carol intercepts the ciphertext $c \equiv m^e \pmod n$

sent by Alice. Also suppose that Bob is willing to sign any *random* message presented by Carol. Explain how Carol can choose a message to be signed by Bob in order to retrieve the secret m. **[H]**

5.17 Let G be a finite cyclic group of order n, and g a generator of G. Suppose that Alice's private and public keys are respectively d and g^d.
(a) Consider a variant of the ElGamal signature scheme, in which s is computed as in Algorithm 5.36, but the roles of d and d' are interchanged in the generation of t, that is, the modified signature $(s, \bar{t}\,)$ on M is generated as:

$$\begin{aligned} s &:= g^{d'}, \\ \bar{t} &:= d^{-1}[H(M) - d'H(s)] \pmod{n}. \end{aligned}$$

Write the verification routine for the modified scheme.
(b) Show that forging modified ElGamal signatures is as difficult as computing discrete logarithms in G. You may assume that a forger can arrange d' of her choice.
(c) Explain why signature generation is (a bit) more efficient in the modified scheme. Suppose that because of this enhanced performance Alice decided to switch to the modified scheme, but for backward compatibility she maintained both the original signature (s, t) and the modified signature $(s, \bar{t}\,)$ on a message M. What went wrong?

5.18 Show that:
(a) There are two valid ECDSA signatures on each message.
(b) There are three valid XTR–DSA signatures on each message.
(Here we call a signature valid, if it passes the verification routine.)

5.19 **(a)** Write the versions with message recovery of the RSA, Rabin, Schnorr and Nyberg–Rueppel signature schemes.
(b) Describe the possibilities of existential forgery for these versions. (Since hash functions cannot be inverted, they are not used for signature schemes with message recovery, and so the problem of existential forgery is more acute in this case. To avoid such forgeries the signer should add some *redundancy* to each message block before signing the same. An existentially forged signature is likely to correspond to a message not containing the redundancy.)

5.20 Design the XTR version of the Nyberg–Rueppel signature scheme with appendix (Section 5.4.5). What are the speed-ups achieved by the signature generation and verification routines of the XTR version over the original NR routines?

5.21 Repeat Exercise 5.20 with the Schnorr digital signature scheme (Section 5.4.4).

5.22 **(a)** Deduce that the determinant of the matrix M_c of Equation (5.9) is

$$D = \det M_c = c^{2p+2} + 18c^{p+1} - 4(c^{3p} + c^3) - 27 \in \mathbb{F}_p\,.$$

(b) Demonstrate that

$$M_c^{-1} = \frac{1}{D} \begin{pmatrix} 2c^2 - 6c^p & 2c^{2p} + 3c - c^{p+2} & c^{p+1} - 9 \\ 2c^{2p} + 3c - c^{p+2} & (c^2 - 2c^p)^{p+1} - 9 & (2c^{2p} + 3c - c^{p+2})^p \\ c^{p+1} - 9 & (2c^{2p} + 3c - c^{p+2})^p & (2c^2 - 6c^p)^p \end{pmatrix}.$$

5.23 Let p, q, p', q' be distinct odd primes with $p = 2p' + 1$ and $q = 2q' + 1$, and let $n := pq$ (as in the RSA-based undeniable signature scheme).
(a) Let $\alpha \in \mathbb{Z}_n^*$. Show that $\mathrm{ord}_n\, \alpha \in \{1, 2, p', q', 2p', 2q', p'q', 2p'q'\}$. **[H]**
(b) Argue that there are exactly four elements in $\mathbb{Z}_n^*$ of order $\leqslant 2$.
(c) Let $\alpha \not\equiv \pm 1 \pmod{n}$ and $\mathrm{ord}_n\, \alpha < p'q'$. Show that $\gcd(\alpha - 1, n)$ or $\gcd(\alpha + 1, n)$ is a non-trivial divisor of n. How many such elements α does $\mathbb{Z}_n^*$ contain?
(d) Let $\gamma \in \mathbb{Z}_n^*$ have order $p'q'$ or $2p'q'$. Show that $\alpha^4 \in \langle \gamma \rangle$ for every $\alpha \in \mathbb{Z}_n^*$.
(e) Look at the denial protocol for the GKR RSA signature scheme (Algorithm 5.61) and assume that $p' < q'$. Suppose that (M, s) is a forged signature (that is, $s \notin \mathrm{Sig}M$) on some message M with $m := H(M) \in \mathbb{Z}_n^*$. Show that $s \equiv \alpha m^d \pmod{n}$ for some $\alpha \in \mathbb{Z}_n^*$ with $\mathrm{ord}_n\, \alpha \geqslant p'$. Deduce that $\mathrm{ord}_n(ms^{-e}) \geqslant p'$. Conclude that if $4k < p'$, then there exists a unique $i' \in \{4, 8, \ldots, 4k\}$ (namely, $i' = i$) for which Congruence (5.11) holds.

5.24 **(a)** Write the shortened versions of ECDSA signature generation and verification.
(b) Write the signcryption and unsigncryption algorithms based on shortened ECDSA.

5.5 Entity Authentication

Entity authentication (also called *identification*) is a process by means of which an entity Alice, called the *claimant*, proves her identity to another entity Bob, called the *prover* or the *verifier*. Alice is assumed to possess some secret piece(s) of information that no intruder is expected to know. During the execution of the identification protocol, an interaction takes place between Alice and Bob. If the interaction allows Bob to conclude (deterministically or with high probability) that the claimer possesses the secret knowledge, he accepts the claimer as Alice. An intruder Carol lacking the secret information is expected (with high probability) to fail to convince Bob of her identity as Alice. This is how entity authentication schemes tend to prevent impersonation attacks by intruders. Typically, identification schemes are used to protect access to some sensitive piece(s) of data, like a user's (or a group's) private files in a computer or an account in a bank. Both secret-key and public-key techniques are used for the realization of entity authentication protocols.

5.5.1 Passwords

A *password* is a small string to be remembered by an entity and produced verbatim to the verifier at the time of identification. The most common example is a computer password used to protect access to a user's private working area in a file system. In this case, an alphanumeric string (or a string that can be input using a computer keyboard) of length between 4 and 20 characters is normally used as the secret information associated with an entity. Passwords are also used to prevent misuse of certain physical objects (like an ATM card for withdrawing cash from one's bank account, a prepaid telephone card) by anybody other than the legitimate owners of the objects. In this case, a password usually consists of a sequence of four to ten digits and is also called a *personal identification number* or a *PIN*.

In order that Bob can recognize an entity from her password, a possibility for Bob is to store the (entity,password) pairs corresponding to all the entities that are expected to participate in identification interactions with Bob. When Alice enters her password, Bob checks if Alice's input is the same as what he stores in the pair for Alice. The file(s) storing these private records should be preserved with high secrecy, and neither read nor write access should be granted to any user. But a privileged user (the *superuser*) is usually given the capability to inspect any file (even read-protected ones) and, therefore, can make misuse of the passwords.

This problem can be avoided by storing, instead of the passwords themselves, a one-way transform of the passwords.[3] When Alice enters a password P, Bob computes the transform $f(P)$ and compares $f(P)$ with the record stored for Alice. The identity of Alice is accepted if and only if a match occurs. The password file now need not be read-protected, since any intruder (even the superuser) knowing the value $f(P)$ cannot easily compute P.

Passwords should be chosen from a space large enough to preclude exhaustive search by an intruder in feasible time. Unfortunately, however, it is a common tendency for human users to choose passwords from limited subsets of the allowed space. For example, use of lower case characters, dictionary words, popular names, birth dates and so on in passwords makes attacks on passwords much easier. A strategy to foil such *dictionary-based attacks* is to use a pseudorandom bit sequence S known as the *salt* and apply the one-way function f to a combination of the password P and the salt S. That is, a function $f(P, S)$ is now stored against an entity Alice having a password P. The combination (P, S) is often referred to as a *key* for the password scheme. Since a password now corresponds to many possible keys, the search space for an intruder increases dramatically. For instance, if S is a pseudorandomly chosen bit string of length 64, the intruder has to compute $f(P, S)$ for a total of 2^{64} times in order to guess the correct candidates for S for each P under trial. It is also necessary that the same key is not chosen for two different entities. If the salt S is a 64-bit string, then by the birthday paradox a collision between two keys is expected to occur only after (at least) 2^{32} keys are generated.

A second strategy to strengthen the protection of passwords is to increase the so-called *iteration count* n, that is, instead of storing $f(P, S)$ for each password P, Bob now stores $f^n(P, S)$. An n-fold application of the function f increases by a factor of n both the time for password verification and for exhaustive search by an intruder. For a legitimate user, this is not really a nuisance, since computation of $f^n(P, S)$ only once during identification is tolerable (and may even be unnoticeable), whereas to an intruder breaking a password simply becomes n times as difficult. In typical applications, values of $n \geqslant 1000$ are recommended.

In some situations, it is advisable to *lock* access to a password-protected area after a predetermined number of (say, three) wrong passwords have been input in succession. This is typically the case with PINs for which the search space is rather small. For unlocking the access (to the legitimate user Alice), a second longer key (again known only to Alice) is used or human intervention is called for.

As a case study, let us briefly describe the password scheme used by the UNIX operating system. During the creation of a password a user supplies a string P of

[3] Informally speaking, a *one-way function* is one which is computationally infeasible to invert.

eight 7-bit ASCII characters as the password. (Longer strings are truncated to first 8 characters.) A 56 bit DES[4] key K is constructed from P. A 12-bit random salt S is obtained from the system clock at the time of the creation of the password. The zero message (that is, a block of 64 zero bits) is then iteratively encrypted $n = 25$ times using K as the key. The encryption algorithm is a variant of the DES, that depends on the salt S. The output ciphertext and the salt (which account for a total of $64 + 12 = 76$ bits) are then packed into eleven 7-bit ASCII characters and stored in the password file (usually /etc/passwd). When UNIX was designed (in 1970), this algorithm, often referred to as the UNIX *crypt* password algorithm, was considered to be reasonably safe under the assumption of the difficulty of finding a DES key from a plaintext–ciphertext pair. With today's hardware and software speed, a motivated attacker can break UNIX passwords in very little time.

Password-based authentication schemes suffer from the disadvantage that the user has to disclose her secret P to the verifier. The verifier may misuse the knowledge of P by storing it secretly and deploying it afterwards. During the process of computation of $f^n(P, S)$ the string P resides in the machine's memory. An eavesdropper capable of monitoring the temporary storage holding the string P easily gets its value. In view of these shortcomings, password schemes are referred to as *weak authentication* schemes.

5.5.2 Challenge–Response Algorithms

In a *strong authentication scheme*, the claimant proves the possession of a secret knowledge to a verifier without disclosing the secret to the verifier. One of the communicating entities generates a random bit string c known as the *challenge* and sends c (or a function of c) to the other. The latter then reacts to the challenge appropriately, for example, by sending a *response* string r to the former. Strong authentication schemes are, therefore, also called *challenge–response authentication* schemes. The communication between the entities depends both on the random challenge and on the secret knowledge of the claimant. An intruder lacking the secret knowledge of a valid claimant cannot take part properly in the interaction. Furthermore, since a random challenge is used during each invocation of the identification protocol, an eavesdropper cannot use the intercepted transcripts of a particular session for a future invocation of the protocol.

Public-key protocols can be used to realize challenge–response schemes. We assume that Alice is the claimant and Bob is the verifier. Without committing to specific algorithms, we denote the public and private keys of Alice by e and d, and the encryption and decryption transforms by f_e and f_d respectively. Alice proves her identity by demonstrating her knowledge of d (but without revealing d) to Bob. Bob uses the transform f_e and Alice the transform f_d under the respective keys e and d. If a key d' other than d is used by Carol in conjunction with e, some step of the interaction detects this and the protocol rejects Carol's claim to be Alice. We describe two challenge–response schemes that differ in the sequence of applying the transforms f_e and f_d.

A challenge–response scheme based on encryption–decryption

In this scheme, Bob (the verifier) first generates a random string r, encrypts the same by the public key of Alice (the claimant) and sends the ciphertext c (the challenge) to Alice.

[4]The *data encryption standard* (DES) is a well-known symmetric-key cipher (Section A.2.1).

Alice uses her private key to decrypt c to the message r' and sends r' (the response) back to Bob. Identification of Alice succeeds if and only if $r = r'$. Algorithm 5.65 illustrates the details of this scheme. It employs a one-way function H (like a hash function) for a reason explained later. This scheme checks whether the claimant can recover the random string r correctly. A knowledge of the decryption key d is needed for that.

Algorithm 5.65 **Challenge–response authentication based on encryption**

Bob generates a random bit string r and computes $w := H(r)$.
Bob reads Alice's (authentic) public key e and computes $c := f_e(r, e)$.
Bob sends (w, c) to Alice.
Alice computes $r' := f_d(c, d)$.
if $(H(r') \neq w)$ { Alice quits the protocol. }
Alice sends r' to Bob.
Bob identifies Alice if and only if $r' = r$.

The string $H(r) = w$ is called the *witness*. By sending w to Alice, Bob convinces her of his knowledge about the secret r without disclosing r itself. If Bob (or a third party pretending to be Bob) tries to cheat, Alice has the option to abort the protocol prematurely. In other words, Alice does not have to decrypt an arbitrary ciphertext presented by Bob without confirming that Bob knows the corresponding plaintext.

A challenge–response scheme based on digital signatures

In the scheme explained in Algorithm 5.66, Alice (the claimant) first does the private key operation, that is, Alice sends her digital signature on a message to Bob (the prover). Bob then verifies the signature of Alice by employing the encryption transform with Alice's public key.

Algorithm 5.66 **Challenge–response authentication based on signature**

Bob selects a random string r_B.
Bob sends r_B to Alice.
Alice selects a random string r_A.
Alice generates the signature $s := f_d(r_A||r_B, d)$.
Alice sends (r_A, s) to Bob.
Bob reads Alice's (authentic) public key e.
Bob retrieves the strings r'_A and r'_B satisfying $r'_A||r'_B = f_e(s, e)$.
Bob identifies Alice if and only if $r'_A = r_A$ and $r'_B = r_B$.

This authentication scheme is based on the assumption that only a person knowing Alice's private key d can generate a signature s that leads to the equalities $r'_A = r_A$ and $r'_B = r_B$. Using only r_A and the signature $s = f_d(r_A, d)$ would demonstrate to Bob that Alice possesses the requisite knowledge of d. The random string r_B is used to prevent the so-called *replay attack*. If r_B were not used, an eavesdropper Carol intercepting the transcripts of a session can later claim her identity as Alice by simply

supplying r_A and Alice's signature on r_A to Bob. Using a new r_B in every session (and incorporating it in the signature) guarantees that the signature varies in different sessions, even when r_A remains the same.

There is an alternative strategy by which the use of the random string r_B can be avoided. All we have to ensure is that a value of r_A used once cannot be reused in a subsequent session. This can be achieved by using a *timestamp*, which is a string reflecting the time when a certain event occurs (in our case, when Alice generates the signature). Thus, if Alice gets the local time t_A, computes the signature $s := f_d(t_A, d)$ and sends (t_A, s) to Bob, it is sufficient for Bob to check that the timestamp t_A is valid. A possible criterion for the validity of Alice's timestamp t_A is that the difference between t_A and the time when Bob is verifying the signature is within an allowed bound (predetermined, based on the approximate time for the communication). But it may be possible for an adversary to provide to Bob the timestamp t_A and Alice's signature on t_A, before t_A expires. Therefore, Bob should additionally ensure that timestamps from Alice come in a strictly ascending order. Maintaining the timestamp for the last interaction with Alice takes care of this requirement. Algorithm 5.67 describes the modified version of Algorithm 5.66, based on timestamps. A problem with timestamps is that (local) clocks across a network have to be properly synchronized.

Algorithm 5.67 **Using timestamp in challenge–response authentication**

Alice reads the local time t_A.
Alice generates the signature $s := f_d(t_A, d)$.
Alice sends (t_A, s) to Bob.
Bob reads Alice's (authentic) public key e.
Bob retrieves the time-stamp $t'_A = f_e(s, e)$.
Bob identifies Alice if and only if $t_A = t'_A$ and this timestamp is valid.

Mutual authentication

So far, we have described identification schemes that are *unidirectional* or *unilateral* in the sense that only Alice tries to prove her identity to Bob. For *mutual* authentication between Alice and Bob, the above schemes can be used a second time by reversing the roles of Alice and Bob. Algorithm 5.68 describes an alternative strategy that achieves mutual authentication with reduced communication overhead (compared to two invocations of the unidirectional scheme). Now, the key pairs (e_A, d_A) and (e_B, d_B) and the transforms $f_{e,A}$, $f_{d,A}$ and $f_{e,B}$, $f_{d,B}$ of both Alice and Bob should be used.

5.5.3 Zero-Knowledge Protocols

The challenge–response schemes described above ensure that the claimant's secret is not made available to the verifier (or a listener to the communication between the verifier and the claimant). But the claimant uses her private key for generating the response and, therefore, it continues to remain possible that a verifier extracts some partial information on the secret by choosing challenges strategically.

Algorithm 5.68 **Mutual authentication**

Bob selects a random string r_B.
Bob sends r_B to Alice.
Alice selects a random string r_A.
Alice generates the signature $s_A := f_{d,A}(r_A||r_B, d_A)$.
Alice sends (r_A, s_A) to Bob.
Bob reads Alice's (authentic) public key e_A.
Bob retrieves the strings r'_A and r'_B satisfying $r'_A||r'_B = f_{e,A}(s_A, e_A)$.
Bob identifies Alice if and only if $r'_A = r_A$ and $r'_B = r_B$.
Bob generates the signature $s_B := f_{d,B}(r_B||r_A, d_B)$.
Bob sends s_B to Alice.
Alice reads Bob's (authentic) public key e_B.
Alice retrieves the strings r''_B and r''_A satisfying $r''_B||r''_A = f_{e,B}(s_B, e_B)$.
Alice identifies Bob if and only if $r''_A = r_A$ and $r''_B = r_B$.

Using a *zero-knowledge* (*ZK*) *protocol* overcomes this difficulty in the sense that (absolutely) no information on the claimant's secret is leaked out during the conversation between the claimant and the verifier. The verifier (or a listener) continues to remain as much ignorant of the secret as he was before the invocation of the protocol. In other words, the verifier (or a listener) does not learn anything form the conversation, that he could not learn by himself in absence of the claimant. The only thing the verifier gains is the confidence whether the claimant actually knows the secret or not. This is intuitively the defining feature of a ZK protocol.

Similar to other public-key techniques, the security of the ZK protocols is based on the intractability of some difficult computational problems. A repeated use of a public-key scheme with a given set of parameters may degrade the security of the scheme under those parameters. For example, each encryption of a message (or each generation of a signature) makes available a plaintext–ciphertext pair which may eventually help a cryptanalyst. A ZK protocol, on the other hand, does not lead to such a degradation of the security of the protocol, irrespective of how many times it is invoked.

We stick to the usual scenario: Alice is the claimant, Bob is the verifier and Carol is an eavesdropper trying to impersonate Alice. In the jargon of ZK protocols, Alice (and not Bob) is called the prover. In order to avoid confusions, we continue to use the terms *claimant* and *verifier*. A ZK protocol is usually a three-pass interactive protocol. To start with, Alice chooses a random *commitment* and sends a *witness* of the commitment to Bob. A new commitment should be selected by Alice during each invocation of the protocol in order to guard against an adversarial verifier. Upon receiving the witness, Bob chooses and sends a random *challenge* to Alice. Finally, Alice replies by sending a *response* to the challenge. If Alice knows the secret (and performs the protocol steps correctly), her response can be easily proved by Bob to be valid. Carol, in an attempt to impersonate Alice without knowing the secret, can produce the valid response with a probability P bounded away from 1. If P happens not to be negligibly small, then the protocol can be repeated a sufficient number of times, so that Carol's probability of giving the correct response on all occasions becomes extremely low.

The parameters and the secrets for a ZK protocol can be set privately by each claimant. Another alternative is that a *trusted third party* (*TTP*) generates a set of parameters and makes these parameters available for use by every claimant over a network. A second duty of the TTP is to register a secret against each entity. The secret may be generated either by the TTP or by the respective entity. The knowledge of this (registered) secret by an entity is equivalent to her identity in the network. Finally, the authenticity of the public key of an entity is ensured by the digital signature of the TTP on the public key. For simplicity, however, we will not bother about the existence of the TTP and the way in which the secret (the possession of which by Alice is to be proved) has been created and/or handed over to Alice. We will also assume that each entity's public key is authentic.

The Feige–Fiat–Shamir (FFS) protocol

The FFS protocol (Algorithm 5.69) is based on the intractability of computing square roots modulo a composite integer n. We take $n = pq$ with two distinct primes p and q each congruent to 3 modulo 4.

Algorithm 5.69 **Feige–Fiat–Shamir zero-knowledge protocol**

Selection of domain parameters:
Select two large distinct primes p and q each congruent to 3 modulo 4.
$n := pq$.
Select a small integer t. /* The probability of a successful cheat is 2^{-t} */

Selection of Alice's secret:
Alice selects t random integers $x_1, \ldots, x_t \in \mathbb{Z}_n^*$.
Alice selects t random bits $\delta_1, \ldots, \delta_t \in \{0, 1\}$.
Alice computes $y_i := (-1)^{\delta_i}(x_i^2)^{-1} \pmod n$ for $i = 1, \ldots, t$.
Alice makes $(y_1, \ldots, y_t)$ public and keeps $(x_1, \ldots, x_t)$ secret.

The protocol:
Alice randomly chooses $c \in \mathbb{Z}_n^*$ and $\gamma \in \{0, 1\}$. /* Commitment */
Alice computes and sends to Bob $w := (-1)^\gamma c^2 \pmod n$. /* Witness */
Bob randomly chooses and sends to Alice $\epsilon_1, \ldots, \epsilon_t \in \{0, 1\}$. /* Challenge */
Alice computes and sends to Bob $r := c \prod_{i=1}^{t} x_i^{\epsilon_i} \pmod n$. /* Response */
Bob computes $w' := r^2 \prod_{i=1}^{t} y_i^{\epsilon_i} \pmod n$.
Bob accepts Alice's identity if and only if $w' \neq 0$ and $w' \equiv \pm w \pmod n$.

It is clear from Algorithm 5.69 that knowing the secret $(x_1, \ldots, x_t)$ allows Alice to let Bob accept her identity (as Alice). The check $w' \neq 0$ in the last line is necessary to preclude the commitment $c = 0$, that makes any claimant succeed irrespective of the availability of the knowledge of the secret.

Now, let us see how an opponent (Carol), without knowing the secret, can succeed in impersonating Alice by taking part in this protocol. To start with, we consider the simple case $t = 1$ (which corresponds to Fiat and Shamir's original scheme). Carol can start the process by generating a random c and γ and computing $w = (-1)^\gamma c^2$. Now, Carol should send the response c or cx_1 depending on whether Bob sends $\epsilon_1 = 0$ or 1. Her capability of sending both correctly is equivalent to her knowledge of

x_1. If Bob sends $\epsilon_1 = 0$, then she can provide the correct response c. Otherwise, Carol can at best select a random response from $\mathbb{Z}_n$, and the probability that this is correct is overwhelmingly low. On the other hand, let Carol choose a random c and $\gamma \in \{0, 1\}$ and send the (improper) witness $\tilde{w} := (-1)^\gamma c^2 y_1^{-1} \pmod{n}$. In that case, Carol can answer the valid response $r = c$, if Bob's challenge is $\epsilon_1 = 1$. Sending the correct response $cx_1^{-1} \pmod{n}$ to the challenge $\epsilon_1 = 0$ now requires knowledge of x_1. Therefore, if ϵ_1 is randomly chosen by Bob (without the prior knowledge of Carol), Carol can successfully respond with probability (very close to) $1/2$. For $t \geqslant 1$, this probability of a cheat by Carol can be easily shown to be (very close to) $1/2^t$ which is negligibly small for $t \geqslant 80$.

In practice, however, t is chosen to be $\mathrm{O}(\ln \ln n)$. It is, therefore, necessary to repeat the protocol t' times, so that the probability of a successful cheat becomes (nearly) $1/2^{tt'}$. Taking $t' = \Theta(\ln n)$ is recommended. It can be shown that these choices for t and t' offer the FFS protocol the desired ZK property. Without going into a proof of this assertion, let us informally explain the ZK property of the FFS protocol. Neither Bob nor a listener to the conversation between Alice and Bob can get any idea of the secret $(x_1, \ldots, x_t)$. Bob gets as a response the product of c and those x_i's for which $\epsilon_i = 1$. Since c is randomly chosen by Alice and is not available to Bob, there is no way to choose a strategic challenge. However, if the square root of w (or $-w$) can be computed by Bob, then the interaction may give away partial information on the secret. For example, if Bob chooses the challenge $(\epsilon_1, \epsilon_2, \ldots, \epsilon_t) = (1, 0, \ldots, 0)$, then Alice's response would be cx_1 from which x_1 can be computed by Bob, if he knows c. Thus, the security and the ZK property of the FFS protocol are based on the assumption that computing square roots modulo n is an infeasible computational problem.

The Guillou–Quisquater (GQ) protocol

The GQ identification protocol is based on the intractability of the RSA problem. The correctness of Algorithm 5.70 (for a legitimate claimant) is easy to establish. The check $w' \neq 0$ is necessary to avoid the commitment $c = 0$, which makes a claimant succeed always.

A TTP typically selects the domain parameters p, q, n, e and d. It also selects m and gives s to Alice without revealing d. The execution of the protocol does not require the use of the decryption exponent d. In fact, d is a global secret, whereas s is Alice's personal secret. Alice tries to prove the knowledge of s (and not of d).

In the GQ algorithm, the power s^ϵ is blinded by multiplying it with the random commitment c. As a witness for c, Alice presents its encrypted version w. With the assumption that RSA decryption without the knowledge of the decryption exponent d is infeasible, Bob (or an eavesdropper) cannot compute c and hence cannot separate out the value of s^ϵ. Thus, no partial information on s is provided. Furthermore, each invocation requires a random ϵ. In order to compute a strategic witness, Carol can at best have a guess of ϵ. The guess is correct with a probability of $1/e$. If e is reasonably large, the probability of a successful cheat is low. However, larger values of e lead to more expensive generation of the witness from the commitment (and also of the response). So small values of e (say, $2^{16} + 1 = 65{,}537$) are usually recommended. In that case, repeating the protocol a suitable number of times makes Carol's chance of cheating as small as one desires. Taking $t'e$ (where t' is the number of iterations of the

Algorithm 5.70 **Guillou–Quisquater zero-knowledge protocol**

Selection of domain parameters:
Select two distinct large primes p and q and set the modulus $n := pq$.
Select an exponent $e \in \mathbb{Z}_n^*$ and compute $d := e^{-1} \pmod{\phi(n)}$.
The pair (n, e) is made public and d is kept secret.

Selection of Alice's secret:
Alice selects a random $m \in \mathbb{Z}_n^*$ and computes $s := m^{-d} \pmod{n}$.
Alice makes m public and keeps s secret.

The protocol:
Alice selects a random $c \in \mathbb{Z}_n^*$. /* Commitment */
Alice computes and sends to Bob $w := c^e \pmod{n}$. /* Witness */
Bob selects and sends to Alice a random $\epsilon \in \{1, \ldots, e\}$. /* Challenge */
Alice computes and sends to Bob $r := cs^\epsilon \pmod{n}$. /* Response */
Bob computes $w' := m^\epsilon r^e \pmod{n}$.
Bob accepts Alice's identity if and only if $w' \neq 0$ and $w' = w$.

protocol) of the order of $(\log n)^\alpha$ for some constant α gives the GQ protocol the desired zero-knowledge property.

The Schnorr protocol

The Schnorr protocol is based on the intractability of computing discrete logarithms in a large prime field $\mathbb{F}_p$. We assume that a suitably large prime divisor q of $p - 1$ and an element $g \in \mathbb{F}_p^*$ of multiplicative order q are known. The algorithm works in the subgroup of $\mathbb{F}_p^*$, generated by g. In order to make the known algorithms for solving the DLP infeasible for the field $\mathbb{F}_p$, one should have $q > 2^{160}$.

Algorithm 5.71 **Schnorr zero-knowledge protocol**

Selection of domain parameters:
Select a large prime p such that $p - 1$ has a large prime divisor q.
Select an element $g \in \mathbb{F}_p^*$ having multiplicative order q modulo p.
Publish (p, q, g).
Select a small integer $t < \lg q$. /* The probability of a successful cheat is 2^{-t} */

Selection of Alice's secret:
Alice chooses a random secret integer $d \in \{2, \ldots, q - 1\}$.
Alice computes and makes public the integer $y := g^{-d} \pmod{p}$.

The protocol:
Alice chooses a random $c \in \{1, \ldots, q - 1\}$. /* Commitment */
Alice computes and sends to Bob $w := g^c \pmod{p}$. /* Witness */
Bob selects and sends to Alice a random $\epsilon \in \{1, \ldots, 2^t\}$. /* Challenge */
Alice computes and sends to Bob $r := d\epsilon + c \pmod{q}$. /* Response */
Bob computes $w' := g^r y^\epsilon \pmod{p}$.
Bob accepts Alice's identity if and only if $w' = w$.

We leave the analysis of correctness and security of this protocol to the reader. The secret s is masked from Bob and other eavesdroppers by introducing the random additive bias c modulo q. The probability of a successful cheat by an adversary is 2^{-t}, since ϵ is chosen randomly from a set of cardinality 2^t. Usually the Schnorr protocol is not used iteratively. Therefore, $t \geqslant 40$ is recommended for making the probability of cheating negligible. On the other hand, if t is too large, then the protocol can be shown to lose the ZK property. For the generation of the witness from the commitment, Alice computes a modular exponentiation to an exponent which is $\mathrm{O}(q)$. Generating the response, on the other hand, involves a single multiplication (and a single addition) modulo q and hence is very fast.

EXERCISE SET 5.5

5.25 **(a)** Describe how a zero-knowledge witness–challenge–response identification scheme can be converted to a signature scheme. **[H]**
(b) Write the Feige–Fiat–Shamir, Guillou–Quisquater and Schnorr signature schemes based on the corresponding identification schemes.

5.26 Let $n := pq$ with distinct primes p and q each congruent to 3 modulo 4.
(a) Show that -1 is a quadratic non-residue modulo p and modulo q.
(b) If $a \in \mathbb{Z}_n^*$ is a quadratic residue modulo n, prove that a has exactly four square roots modulo n, of which exactly one is a quadratic residue modulo n.
(c) Consider the following identification protocol in which Alice wants to prove to Bob her knowledge of the factorization of $n = pq$. Assume that p and q are sufficiently large so that computing square roots modulo n is infeasible without the knowledge of the factorization of n. Argue that Alice can prove her identity to Bob if and only if she knows the factorization of n.

A bad zero-knowledge protocol

Bob chooses a random $x \in \mathbb{Z}_n$ and computes $a := x^4 \pmod{n}$.
Bob sends a to Alice.
Alice computes four square roots of a modulo n and picks up the unique square root b which is a quadratic residue modulo n.
Alice sends b to Bob.
Bob accepts Alice's claim if and only if $b \equiv x^2 \pmod{n}$.

(d) Conclude that this is not a good zero-knowledge protocol, by demonstrating that Bob can maliciously send a bad a to Alice so that during the execution of the protocol he gathers enough information to factor n. **[H]**

Chapter Summary

All the material studied in earlier chapters culminates in this relatively short chapter which describes some popular cryptographic algorithms. We address most of the problems relevant in cryptography, namely, encryption, key agreement, digital signatures and entity authentication. Against each algorithm we mention the (provable or alleged) source of security of the algorithm.

Encryption algorithms are treated first. We start with the seemingly most popular *RSA algorithm*. This algorithm derives its security from the RSA key inversion problem and the RSA problem. The key inversion problem is probabilistic polynomial-time equivalent to the integer factorization problem. The intractability of the RSA problem is unknown. At present no algorithm other than factoring the RSA modulus is known for solving the RSA problem. We subsequently describe *Rabin encryption* (based on the square root problem), *Goldwasser–Micali encryption* (based on the quadratic residuosity problem), *Blum–Goldwasser encryption* (based on the square root problem), *ElGamal encryption* (based on the Diffie–Hellman problem) and *Chor–Rivest encryption* (based on a variant of the subset sum problem). The *XTR encryption* algorithm is essentially an efficient implementation of ElGamal encryption and is based on a tricky representation of elements in certain finite fields. The last encryption algorithm we discuss is the *NTRU algorithm*. It derives its security from a mixing system that uses the algebra $\mathbb{Z}[X]/\langle X^n - 1\rangle$. Attacks on NTRU based on the shortest vector problem are also known.

The basic key-agreement scheme is the *Diffie–Hellman scheme*. In order to prevent *small-subgroup attacks* on this scheme, one employs a technique known as *cofactor expansion*. We then explain *unknown key-share attacks* against key-agreement schemes. These attacks necessitate the use of *authenticated key agreement* schemes. The *MQV algorithm* is presented as an example of an authenticated key-agreement scheme.

Next come digital signature algorithms. Digital signatures may be classified in two broad categories: *signature schemes with appendix* and *signature schemes with message recovery*. In this book, we study only the signature schemes with appendix. As specific examples of signature schemes, we first explain *RSA* and *Rabin signatures*. Then, we present several variants of discrete-log-based signature schemes: *ElGamal signatures*, *Schnorr signatures*, *Nyberg–Rueppel signatures*, the *digital signature algorithm* (*DSA*) and its elliptic curve variant *ECDSA*. All the discrete-log (over finite fields)-based signature schemes have efficient *XTR* implementations. The *NTRUSign* algorithm is the last general-purpose signature scheme discussed in this section.

We then present a treatment of some special signature schemes. *Blind signatures* are created on messages unknown to the signer. Three blind signature schemes are described: *Chaum*, *Schnorr* and *Okamoto–Schnorr schemes*. An *undeniable signature*, on the other hand, requires an active participation of the signer at the time of verification and comes with a *denial protocol* that prevents a signer from denying a valid signature at a later time. The *Chaum–Van Antwerpen undeniable signature scheme* is based on the discrete-log problem, whereas the *GKR scheme* is based on the RSA problem.

A way to guarantee both authentication and confidentiality of a message is to sign the message and then encrypt the signed message. This involves two basic operations

(signature generation and encryption). Zheng's *signcryption* scheme combines these two primitives with a view to reducing both running time and message expansion.

The final topic we discuss in this chapter is *entity authentication*, a mechanism by means of which an entity can prove its identity to another. Here identity of an entity is considered synonymous with the possession of some secret information by the entity. *Passwords* are called *weak authentication schemes*, since the claimant has to disclose the secret straightaway to the verifier. A *strong authentication scheme* (also called a *challenge–response scheme*) does not reveal the secret to the verifier. We describe two strong authentication schemes; the first is based on encryption and the second on digital signatures. A way to establish mutual authentication between two entities is also presented. Challenge–response algorithms may be vulnerable to some attacks mounted by the verifier. A *zero-knowledge protocol* comes with a proof that during the authentication conversation no information is leaked to the verifier. Three zero-knowledge protocols are discussed: the *Feige–Fiat–Shamir protocol*, the *Guillou–Quisquater protocol*, and the *Schnorr protocol*.

Suggestions for Further Reading

Public-key cryptography was born from the seminal works of Diffie and Hellman [78] and Rivest, Shamir and Adleman [252]. Though still young, this area has induced much research in the last three decades. In this chapter, we have made an attempt to summarize some important cryptographic algorithms proposed in the literature. The original papers where these techniques have been introduced are listed below. We don't plan to be exhaustive, but mention only the most relevant resources.

Algorithm	Reference(s)
RSA encryption	[252]
Rabin encryption	[246]
Goldwasser–Micali encryption	[117]
Blum–Goldwasser encryption	[27]
ElGamal encryption	[84]
Chor–Rivest encryption	[54]
XTR encryption	[170, 172, 171, 173, 289, 297]
NTRU encryption	[130]
Identity-based encryption	[267, 34, 35]
Diffie–Hellman key exchange	[78]
Menezes–Qu–Vanstone key exchange	[161]
RSA signature	[252]
Rabin signature	[246]
ElGamal signature	[84]
Schnorr signature	[263]
Nyberg–Rueppel signature	[223, 224]
DSA	[220]
ECDSA	[141]
XTR signature	[170, 172, 171, 173, 289, 297]
NTRUSign	[110, 111, 128, 129, 131, 217]
Chaum blind signature	[48, 49, 50]
Schnorr blind signature	[263, 202]
Okamoto–Schnorr blind signature	[227, 236]
Chaum–Van Antwerpen undeniable signature	[51, 52, 53]
RSA undeniable signature	[109, 187, 102, 186]
Signcryption	[310, 311, 312]
Signcryption based on elliptic curves	[313, 314]
Identity-based signcryption	[178, 185]
Feige–Fiat–Shamir ZK protocol	[90, 91]

Guillou–Quisquater ZK protocol	[122]
Schnorr ZK protocol	[263]

The *Handbook of Applied Cryptography* [194] is a single resource where most of the above algorithms have been discussed in good details. See Chapter 8 of this book for encryption algorithms, Chapter 11 for digital signatures and Chapter 10 for identification schemes.

There are several other (allegedly) intractable mathematical problems based on which cryptographic protocols can be built. Some of the promising candidates that we left out in the text are summarized below:

Algorithm	**Intractable problem**
LUC [284, 285, 286]	RSA and ElGamal-like problems based on Lucas sequences
Goldreich–Goldwasser–Halevi [115]	lattice-basis reduction
Patarin's hidden field equation (HFE) [232]	solving multivariate polynomial equations
EPOC/ESIGN [97, 228]	factorization of integers p^2q
McEliece encryption [190]	decoding of error-correcting codes
Number field cryptography [38, 39]	discrete log problem in class groups of quadratic fields
KLCHKP (Braid group cryptosystem) [148]	Braid conjugacy problem

The Internet site http://www.tcs.hut.fi/∼helger/crypto/link/public/index.html is a good place to start, for more information on these (and some other) cryptosystems. Also visit http://www.kisa.or.kr/technology/sub1/index-PKC.htm.

The obvious question that crops up now is that, given so many different cryptographic schemes, which one a user should go for.[5] There is no clear-cut answer to this question. One has to study the relative merits and demerits of the systems. If computational efficiency is what matters, we advocate users to go for NTRU schemes. Having said that, we must also add that the NTRU scheme is relatively new and has not yet

[5]It is worthwhile to issue a warning to the readers. Many cryptographic algorithms (and also the idea of public-key cryptography) are/were patented. In order to implement these algorithms (in particular, for commercial purposes), one should take care of the relevant legal issues. We summarize here some of the important patents in this area. The list is far from exhaustive.

Patent No.	**Covers**	**Patent holder**	**Date of issue**
US 4,200,770	Diffie–Hellman key exchange (includes ElGamal encryption)	Stanford University	Apr 29, 1980
US 4,218,582	Public-key cryptography	Stanford University	Aug 19, 1980
US 4,405,829	RSA	MIT	Sep 20, 1983
US 5,231,668	DSA	USA, Secretary of Commerce	Jul 27, 1993
US 5,351,298	LUC	P. J. Smith	Sep 27, 1994
US 5,790,675	HFE	CP8 Transac (France)	Aug 4, 1998
EP 0963635A1 / WO 09836526	XTR	Citibank (North America)	Dec 15, 1999 Aug 20, 1998
US 6,081,597	NTRU	NTRU Cryptosystems, Inc.	Jun 27, 2000
—	EPOC/ESIGN	Nippon Telegraph and Telephone Corporation	Apr 17, 2001

withstood sufficient cryptanalytic attacks. Various attacks on NSS and NTRUSign cast doubt about the practical safety of applying such young schemes in serious applications.

Our mathematical trapdoors are not provably secure and this is where the problems begin. We have to rely on historical evidences that should not be collected too hastily. Slow as it is, RSA has stood the test of time, and has successfully survived more than twenty years of cryptanalytic attacks [29]. The risks attached to the fact that an unforeseen attack will break the system tomorrow, appear much less with RSA, compared to newer schemes that have enjoyed only little cryptanalytic studies. The hidden monomial system proposed by Imai and Matsumoto [188] was broken by Patarin [231]. As a by-product, Patarin came up with the idea of cryptosystems based on hidden field equations (HFE) [232]. No serious attacks on HFE are known till date, but as we mentioned earlier, only time will show whether HFE is going to survive.

Bruce Schneier asserts in his Crypto-gram news-letter (15 March 1999, http://www.counterpane.com/crypto-gram.html): *No one can duplicate the confidence that RSA offers after 20 years of cryptanalytic review. A standard security review, even by competent cryptographers, can only prove insecurity; it can never prove security. By following the pack you can leverage the cryptanalytic expertise of the worldwide community, not just a handful of hours of a consultant's time.*

Twenty-odd years is definitely not a wide span of time in the history of evolution of our knowledge, but public-key cryptography is only as old as RSA is!

Standards

6.1 Introduction
6.2 IEEE Standards
6.3 RSA Standards
Chapter Summary
Sugestions for Further Reading

6

In theory, there is no difference between theory and practice.
But, in practice, there is.
— Jan L. A. van de Snepscheut

ECC curves are divided into three groups, weak curves,
inefficient curves, and curves patented by Certicom.
— Peter Gutmann

Acceptance of prevailing standards often means
we have no standards of our own.
— Jean Toomer (1894 –1967)

6.1 Introduction

Public-key cryptographic protocols deal with sets like the ring $\mathbb{Z}_n$ of integers modulo n, the multiplicative group $\mathbb{F}_q^*$ of units in a finite field or the group $E(\mathbb{F}_q)$ of points in an elliptic curve over a finite field. Messages that need to be encrypted or signed are, on the other hand, usually human-readable text or numbers or keys of secret-key cryptographic protocols, which are typically represented in computers in the form of sequences of bits (or bytes). It is necessary to convert such bit stings (or byte strings) to mathematical elements before the cryptographic algorithms are applied. This conversion is referred to as *encoding*. The reverse transition, that is, converting mathematical entities back to bit strings is called *decoding*.

If Alice and Bob were the only two parties involved in deploying public-key protocols, they could have agreed upon a set of private (not necessarily secret) encoding and decoding rules. In practice, however, when many entities interact over a public network, it is impractical, if not impossible, to have an individual encoding scheme for every pair of communicating parties. This is also unnecessary, because the security of the protocols comes from the encryption process and not from encoding. On the contrary, poorly designed encoding schemes may endanger the security of the underlying protocols.

We, therefore, need a set of standard ways of converting data between various logical formats. This promotes interoperability, removes ambiguities, facilitates simplicity in handling cryptographic data and thereby enhances the applicability and acceptability of public-key algorithms. IEEE (The Institute of Electrical and Electronics Engineers, Inc., pronounced eye-triple-e) and the RSA laboratories have published extensive documents standardizing data conversion and encoding for many popular public-key cryptosystems. Here we summarize the contents of some of these documents. This exposition is meant mostly for software engineers intending to develop cryptographic tool-kits that conform to the accepted standards.

6.2 IEEE Standards

In this section, we outline the first three of the drafts from IEEE, shown in Table 6.1. At the time of writing this book, these are the latest versions of the drafts available

Table 6.1 IEEE drafts on public-key cryptography

Draft	Date	Description
P1363 / D13	12 November 1999	Traditional public-key cryptography based on IFP, DLP and ECDLP
P1363a/D12	16 July 2003	Additional techniques on traditional public-key cryptography
P1363.1/D4	7 March 2002	Lattice-based cryptography
P1363.2/D15	25 May 2004	Password-based authentication
P1363.3/D1	May 2008	Identity-based public-key cryptography

from IEEE. In future, these may be superseded by newer documents. We urge the reader to visit the web-site http://grouper.ieee.org/groups/1363/ for more up-to-date information. Also see the standard *IEEE 1363–2000: Standards Specifications for Public-key Cryptography* [134].

6.2.1 The Data Types

Public-key protocols operate on data of various types. The IEEE drafts specify only the logical descriptions of these data types. The realizations of these data types should be taken care of by individual implementations and are left unspecified.

Bit strings

A *bit string* is a finite ordered sequence $a_0a_1 \ldots a_{l-1}$ of bits, where each bit a_i can assume the value 0 or 1. The length of the bit string $a_0a_1 \ldots a_{l-1}$ is l. The bit a_0 in the bit string $a_0a_1 \ldots a_{l-1}$ is called the *leftmost* or the *first* or the *leading* or the *most significant bit*, whereas the bit a_{l-1} is called the *rightmost* or the *last* or the *trailing* or the *least significant bit*.

The order of appearance of the bits in a bit string is important, rather than the way the bits are indexed or named. That is to say, the most and least significant bits in a given bit string are uniquely determined by their positions of occurrences in the string, and not by the way the individual bits in the string are numbered. Thus, for example, if we call the bit string 01101 as $a_0a_1a_2a_3a_4$, then the leading and trailing bits are a_0 and a_4 respectively. If we index the bits in the same bit string as $a_2a_3a_5a_7a_{11}$, the first bit is a_2 and the last bit is a_{11}. Finally, for the indexing $a_5a_4a_3a_2a_1$, the leftmost and rightmost bits are a_5 and a_1 respectively.

Octet strings

Though bits are the basic building blocks in computer memory, programs typically access memory in groups of 8 bits, known as *octets*. Thus, an octet is a bit string of length 8 and can have one of the 256 values $0000\ 0000$ through $1111\ 1111$. It is convenient to write an octet as a concatenation of two hexadecimal digits, the first (resp. second) one corresponding to the first (resp. last) 4 bits in the octet being treated as an 8-bit integer in base 2. For example, the octet $0010\ 1011$ is represented by $2\mathrm{b}$. It is also often customary to treat an octet $a_0a_1 \ldots a_7$ as the integer (between 0 and 255, both inclusive) whose binary representation is $a_0a_1 \ldots a_7$.

An *octet string* is a finite ordered sequence of octets. The length of an octet string is the number of octets in the string. The leftmost (or first or leading or most significant) and the rightmost (or last or trailing or least significant) octets in an octet string are defined analogously as in the case of bit strings. These octets are dependent solely on their positions in the octet string and are independent of how the individual octets in the octet string are numbered.

Integers

Integers are the whole numbers $0, \pm 1, \pm 2, \ldots$. For cryptographic applications, one typically considers only non-negative integers. Integers used in cryptography may have binary representations requiring as many as several thousand bits.

Prime finite fields

Let p be a prime (typically, odd). The elements of $\mathbb{F}_p$ are represented as integers $0, 1, \ldots, p-1$ under the standard way of associating the integer $a \in \{0, 1, \ldots, p-1\}$ with the congruence class $[a]_p$ in $\mathbb{F}_p = \mathbb{Z}_p = \mathbb{Z}/p\mathbb{Z}$. Arithmetic operations in $\mathbb{F}_p$ are the corresponding integer operations modulo the prime p.

Finite fields of characteristic 2

The elements of the field $\mathbb{F}_{2^m}$ are represented as bit strings of length m. In order to provide the mathematical interpretation of these bit strings, we recall that $\mathbb{F}_{2^m}$ is an m-dimensional $\mathbb{F}_2$-vector space. Let $\beta_0, \ldots, \beta_{m-1}$ be an *ordered* basis of $\mathbb{F}_{2^m}$ over $\mathbb{F}_2$. The bit string $a_0 \ldots a_{m-1}$ is to be identified with the element $a_0\beta_0 + \cdots + a_{m-1}\beta_{m-1}$, where the bit a_i represents the element $[a_i]_2$ of $\mathbb{F}_2$. Selection of the basis $\beta_0, \ldots, \beta_{m-1}$ renders a complete meaning to this representation and determines how arithmetic operations on these elements are to be performed. The following two cases are recommended.

For the *polynomial-basis representation*, one chooses an irreducible polynomial $f(X) \in \mathbb{F}_2[X]$ of degree m and represents $\mathbb{F}_{2^m}$ as $\mathbb{F}_2[X]/\langle f(X)\rangle$. Letting x denote the canonical image of X in $\mathbb{F}_2[X]/\langle f(X)\rangle$ one chooses the ordered basis $\beta_0 = x^{m-1}, \beta_1 = x^{m-2}, \ldots, \beta_{m-1} = 1$. Arithmetic operations in $\mathbb{F}_{2^m}$ under this representation are those of $\mathbb{F}_2[X]$ followed by reduction modulo the defining polynomial $f(X)$. Choice of the irreducible polynomial $f(X)$ is left unspecified in the IEEE drafts.

For the *normal-basis representation*, one selects an element $\theta \in \mathbb{F}_{2^m}$ which is normal over $\mathbb{F}_2$ (see Definition 2.60, p86), and takes the ordered basis $\beta_0 = \theta = \theta^{2^0}, \beta_1 = \theta^{2^1}, \beta_2 = \theta^{2^2}, \ldots, \beta_{m-1} = \theta^{2^{m-1}}$. Arithmetic in $\mathbb{F}_{2^m}$ is carried out as explained in Section 2.9.3.

The IEEE draft P1363a also specifies a *composite-basis representation* of elements of $\mathbb{F}_{2^m}$, provided that m is composite. Let $m = ds$ with $1 < d < m$. One chooses an (ordered) polynomial or normal basis $\gamma_0, \gamma_1, \ldots, \gamma_{s-1}$ of $\mathbb{F}_{2^m}$ over $\mathbb{F}_{2^d}$. An element of $\mathbb{F}_{2^m}$ is of the form $a_0\gamma_0 + a_1\gamma_1 + \cdots + a_{s-1}\gamma_{s-1}$ and is represented by $a_0a_1 \ldots a_{s-1}$, where each a_i, being an element of $\mathbb{F}_{2^d}$, is represented by a bit string of length d. The interpretation of the representation of a_i is dependent on how $\mathbb{F}_{2^d}$ is represented. One can use a polynomial- or normal-basis representation of $\mathbb{F}_{2^d}$ (over $\mathbb{F}_2$), or even a composite-basis representation of $\mathbb{F}_{2^d}$ over $\mathbb{F}_{2^{d'}}$, if d happens to be composite with a non-trivial divisor d'.

Extension fields of odd characteristics

A non-prime finite field of odd characteristic is one with cardinality p^m for some odd prime p and for some $m \in \mathbb{N}$, $m > 1$. The field $\mathbb{F}_{p^m}$ is represented as $\mathbb{F}_p[X]/\langle f(X)\rangle$,

where $f(X) \in \mathbb{F}_p[X]$ is an irreducible polynomial of degree m. An element of $\mathbb{F}_{p^m}$ is then of the form $\alpha = a_{m-1}x^{m-1} + \cdots + a_1 x + a_0$, where $x := X + \langle f(X) \rangle$ and where each a_i is an element of $\mathbb{F}_p$, that is, an integer in the range $0, 1, \ldots, p-1$. The element α is represented as an *integer* by substituting p for x, that is, as the integer $a_{m-1}p^{m-1} + \cdots + a_1 p + a_0 \in \{0, 1, \ldots, p^m - 1\}$ (see the packed representation of Exercise 3.39). In order to interpret an integer between 0 and $p^m - 1$ as an element of $\mathbb{F}_{p^m}$, one has to expand the integer in base p.

*** Elliptic curves**

An elliptic curve defined over a finite field $\mathbb{F}_q$ is specified by two elements $a, b \in \mathbb{F}_q$. Depending on the characteristic of $\mathbb{F}_q$ this pair defines the following curves.

If $\mathrm{char}\,\mathbb{F}_q \neq 2, 3$, then $4a^3 + 27b^2$ must be non-zero in $\mathbb{F}_q$ and the equation of the elliptic curve is taken to be $Y^2 = X^3 + aX + b$.

For $\mathrm{char}\,\mathbb{F}_q = 2$, we must have $b \neq 0$ in $\mathbb{F}_q$ and we use the non-supersingular curve $Y^2 + XY = X^3 + aX^2 + b$. Because of the MOV attack (Section 4.5.1), supersingular curves are not recommended for cryptographic applications.

Finally, if $\mathbb{F}_q$ has characteristic 3, then both a and b must be non-zero in $\mathbb{F}_q$ and the elliptic curve $Y^2 = X^3 + aX^2 + b$ is specified by (a, b).

*** Elliptic curve points**

A point $P = (h.k) \neq \mathcal{O}$ on an elliptic curve defined over $\mathbb{F}_q$ can be represented either in *compressed* or in *uncompressed* form. In the uncompressed form, one represents P as the pair (h, k) of elements of $\mathbb{F}_q$. The compressed form can be either *lossy* or *lossless*. In the lossy compressed form, P is represented by its X-coordinate h only. Such a representation is not unique in the sense that there can be two points on the elliptic curve with the same X-coordinate h. In applications where Y-coordinates of elliptic curve points are not utilized, such a representation can be used. In the lossless compressed form, one represents P as $(h, \hat{k}) \in \mathbb{F}_q \times \{0, 1\}$. There are two solutions (perhaps repeated) for Y for a given value h of X. The bit $\hat{k}$ specifies which of these two values is represented. Depending on how the bit $\hat{k}$ is computed, we have two different lossless compressed forms.

The *LSB compressed form* is applicable for odd prime fields $\mathbb{F}_p$ or fields $\mathbb{F}_{2^m}$ of even characteristic. For $\mathbb{F}_p$, the bit $\hat{k}$ is taken to be the least significant (that is, rightmost) bit of k (treated as an integer). For $\mathbb{F}_{2^m}$, we have $\hat{k} = 0$, if $h = 0$, whereas if $h \neq 0$, then $\hat{k}$ is the least significant bit of the element kh^{-1} treated as an integer via the FE2I conversion primitive described in Section 6.2.2.

The *SORT compressed form* is used for $q = p^m$, $m > 1$. Let $P' = (h, k')$ be the opposite of $P = (h, k)$, that is, $k' = \begin{cases} -k & \text{if } p \text{ is odd,} \\ h + k & \text{if } p = 2. \end{cases}$ One converts k and k' to integers κ and κ' using the FE2I primitive and sets $\hat{k} = \begin{cases} 1 & \text{if } \kappa > \kappa', \\ 0 & \text{otherwise.} \end{cases}$

One may also go for a *hybrid representation* of the elliptic curve point $P = (h, k)$, in which information for both the compressed and the uncompressed representations

for P are stored, that is, P is stored as $(h, k, \hat{k})$ with $\hat{k}$ computed by one of the methods (LSB or SORT) described above.

*** Convolution polynomial rings**

For NTRU public-key cryptosystems, we work in the ring $R := \mathbb{Z}[X]/\langle X^n - 1\rangle$. We denote $x := X + \langle X^n - 1\rangle \in R$ as usual. An element of R is a polynomial $a(x) = a_0 + a_1x + a_2x^2 + \cdots + a_{n-1}x^{n-1}$ with $a_i \in \mathbb{Z}$, and is represented by the ordered n-tuple of integers $(a_0, a_1, \ldots, a_{n-1})$. Addition (resp. subtraction) in R is simply component-wise addition (resp. subtraction), whereas multiplication of $a(x) = a_0 + a_1x + \cdots + a_{n-1}x^{n-1}$ and $b(x) = b_0 + b_1x + \cdots + b_{n-1}x^{n-1}$ gives $c(x) = c_0 + c_1x + \cdots + c_{n-1}x^{n-1}$, where $c_i = \sum_{\substack{0 \leqslant j,k \leqslant n-1 \\ j+k \equiv i \pmod n}} a_jb_k$ (see Section 5.2.8). The IEEE draft P1363.1 designates elements of R as *ring elements*.

It is customary to deal with polynomials in R with small coefficients. If all the coefficients of $a(x) = a_0 + a_1x + \cdots + a_{n-1}x^{n-1} \in R$ are known to be from $\{0, 1\}$, it is convenient to represent $a(x)$ as the bit string $a_0a_1 \ldots a_{n-1}$ instead of as an n-tuple of integers. In this case, $a(x)$ is called a *binary ring element* or simply a *binary element*.

6.2.2 Conversion Among Data Types

The IEEE drafts P1363 and P1363.1 specify algorithms for converting data among the formats discussed above. The standardized data conversion primitives are summarized in Figure 6.1. Though these drafts support elliptic curve cryptography, it is not specified how data representing elliptic curves can be converted to data of other types (like octet strings and bit strings).

We now provide a brief description of the data conversion primitives at a logical level. The implementation details depend on the representations of the data types and are left out here.

Converting bit strings to octet strings (BS2OS)

A bit string $a_0a_1 \ldots a_{l-1}$ can be broken up in groups of eight bits and packed into octets. But we run with difficulty, if the length of the input bit string is not an integral multiple of 8. We have to add extra bits in order the make the length of the augmented bit string an integral multiple of 8. This can be done is several ways and in this context a standard convention needs to be adopted. The IEEE drafts prescribe the following rules:

1. Every extra bit added must be the *zero* bit.
2. Add the *minimal* number of extra bits.
3. Add the extra bits, if any, to the *left*.[1]

[1] At the time of writing this book there is a serious conflict between the latest drafts of P1363 and P1363.1 from IEEE. The former asks to add extra bits to the left, the latter to the right. One of the authors of this book raised this issue in the discussion group stds-p1363-discuss maintained by IEEE and was notified that in the next version of the P1363.1 document this conflict would be resolved in favour of P1363.

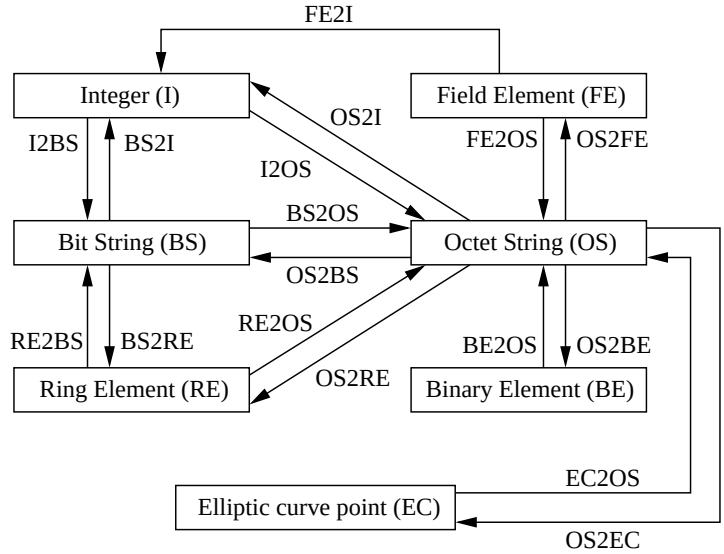


Figure 6.1 IEEE P1363 data types and conversions

In order to see what these rules mean, let $a_0a_1 \ldots a_{l-1}$ be a bit string of length l to be converted to the octet string $A_0A_1 \ldots A_{d-1}$. The length of the output octet string must be $d = \lceil l/8 \rceil$. $8d - l$ zero bits should be added to the left of the input bit string in order to create the augmented bit string $0 \ldots 0a_0a_1 \ldots a_{l-1}$ whose length is $8d$. Now, we start from the left and pack blocks of consecutive eight bits in $A_0, A_1, \ldots, A_{d-1}$. Thus, we have $A_0 = 0 \ldots 0a_0 \ldots a_{k-1}$, $A_1 = a_k \ldots a_{k+7}$, $\ldots$, $A_{d-1} = a_{k+8(d-2)} \ldots a_{k+8(d-2)+7}$, where $k = 8 - (8d - l)$. Note that if l is already a multiple of 8, then $8d - l = 0$, that is, no extra bits need to be added.

As an example, consider the input bit string $01110\ 01101011$ of length 13. The output octet string should be of length $\lceil 13/8 \rceil = 2$. Padding gives the augmented bit string $00001110\ 01101011$. The first octet in the output octet string will then be 00001110, that is, $0\mathrm{e}$; and the second octet will be 01101011, that is, $6\mathrm{b}$.

Converting octet strings to bit strings (OS2BS)

The OS2BS primitive is designed to ensure that if we convert an octet string generated by BS2OS, we should get back the original bit string (that is, the input to BS2OS) with which we started. Suppose that we want to convert an octet string $A_0A_1 \ldots A_{d-1}$. Let us write the bits of A_i as $a_{i,0}a_{i,1} \ldots a_{i,7}$. The desired length l of the output bit string has to be also specified. If $d \neq \lceil l/8 \rceil$, the procedure OS2BS reports error and stops. If $d = \lceil l/8 \rceil$, we consider the bit string

$$a_{0,0}a_{0,1} \ldots a_{0,7}a_{1,0}a_{1,1} \ldots a_{1,7} \ldots a_{d-1,0}a_{d-1,1} \ldots a_{d-1,7}$$

of length $8d$. If the leftmost $8d-l$ bits of this flattened bit string are not all zero, OS2BS should quit after reporting error. Otherwise, the trailing l bits of the flattened bit string is returned.

The reader can check that when $0\mathrm{e}\ 6\mathrm{b}$ and $l = 13$ are input to OS2BS, it returns the bit string $01110\ 01101011$. (See the example in connection with BS2OS.) Notice also that for this input octet string, OS2BS reports error if and only if a value $l \geqslant 17$ or $l \leqslant 11$ is supplied as the desired length of the output bit string.

Converting integers to bit strings (I2BS)

Let a *non-negative* integer n be given. The I2BS primitive outputs a bit string of length l representing n. If $n \geqslant 2^l$, this conversion cannot be done and the primitive reports error and quits. If $n < 2^l$, we write the binary representation of n as

$$n = a_{l-1}2^{l-1} + a_{l-2}2^{l-2} + \cdots + a_1 2 + a_0 \text{ with } a_i \in \{0, 1\}.$$

Treating each a_i as a bit[2], I2BS returns the bit string $a_{l-1}a_{l-2}\ldots a_1 a_0$. One or more leading bits of the binary representation of n may be zero. There is no limit on how many leading zero bits are allowed during the conversion. In particular, the integer 0 gets converted to a sequence of l zero bits for any value of l supplied.

A request to I2BS to convert $n = 2357 = 2^{11} + 2^8 + 2^5 + 2^4 + 2^2 + 2^0$ with $l = 12$ returns $1001\ 00110101$, one with $l = 18$ returns $00\ 00001001\ 00110101$ and finally one with $l \leqslant 11$ reports failure. Note that for neater look we write bit strings in groups of eight and grouping starts from the right. This convention reflects the relationship between bit strings and octet strings, as mentioned above.

Converting bit strings to integers (BS2I)

The primitive BS2I converts the bit string $a_0 a_1 \ldots a_{l-1}$ to the integer $a_0 2^{l-1} + a_1 2^{l-2} + \cdots + a_{l-2}2 + a_{l-1}$, where we again identify a bit with an integer (or a binary digit). As an illustrative example, the bit string $1001\ 00110101$ (or $00\ 00001001\ 00110101$) gets converted to the integer $2^{11} + 2^8 + 2^5 + 2^4 + 2^2 + 2^0 = 2357$. The null bit string (that is, the one of zero length) is converted to the integer 0.

Converting integers to octet strings (I2OS)

In order to convert a *non-negative* integer n to an octet string of length d, we write the base-256 expansion of n as

$$n = A_{d-1}256^{d-1} + A_{d-2}256^{d-2} + \cdots + A_1 256 + A_0,$$

where each $A_i \in \{0, 1, \ldots, 255\}$ and can be naturally identified with an octet. I2OS returns the octet string $A_{d-1}A_{d-2}\ldots A_1 A_0$. Note that the above representation of n to the base 256 is possible if and only if $n < 256^d$. If $n \geqslant 256^d$, I2OS should return failure. Like bit strings, an arbitrary number of leading zero octets are allowed.

Consider the integer $2357 = 9 \times 256 + 53$. The two-digit hexadecimal representations of 9 and 53 are 09 and 35 respectively. Thus, a call of I2OS on this n with $d = 3$ (resp. $d = 2$, resp. $d = 1$) returns $00\ 09\ 35$ (resp. $09\ 35$, resp. failure).

[2]Each a_i is logically an integer which happens to assume one of two possible values: 0 and 1. A bit, on the other hand, is a quantity that can also assume only two possible values. Traditionally, the values of a bit are also denoted by 0 and 1. But one has the liberty to call these values *off* and *on*, or *false* and *true*, or *black* and *white*, or even *armadillo* and *platypus*. To many people, *bit* is an abbreviation for *binary digit* which our a_is logically are. To others, *binit* is a safer and more individualistic acronym for *binary digit*. For I2BS, we identify the two concepts.

Converting octet strings to integers (OS2I)

Let an octet string $A_0A_1 \ldots A_{d-1}$ be given. Each A_i can be identified with a 256-ary digit. OS2I returns the integer $A_0 256^{d-1} + A_1 256^{d-2} + \cdots + A_{d-2}256 + A_{d-1}$. If $d = 0$, the integer 0 should be output.

Converting field elements to octet strings (FE2OS)

In the IEEE P1363 jargon, a field element is an element of the finite field $\mathbb{F}_q$, where q is a prime or an integral power of a prime. We want to convert an element $\beta \in \mathbb{F}_q$ to an octet string. Depending on the value of q, we have two cases:

If $\mathrm{char}\,\mathbb{F}_q$ is odd, β is represented as an integer in $\{0, 1, \ldots, q-1\}$. FE2OS converts β to an octet string of length $\lceil \log_{256} q \rceil$ by calling the primitive I2OS.

If $q = 2^m$, β is represented as a bit string of length m. The primitive BS2OS is called to convert β to an octet string.

Converting octet strings to field elements (OS2FE)

Assume that an octet string is to be converted to an element of the finite field $\mathbb{F}_q$. Again we have two possibilities depending on q.

If $\mathbb{F}_q$ is of odd characteristic, the primitive OS2I is called to convert the given octet string to an integer. This integer is returned as the field element.

If $q = 2^m$, one calls the primitive OS2BS with the given octet string and with the length m supplied as inputs. The resulting bit string is returned by OS2FE. If OS2BS reports error, so should do OS2FE too.

Converting field elements to integers (FE2I)

Let $\beta \in \mathbb{F}_q$ and the integer equivalent of β be sought for. If q is odd, then β is already represented as an integer (in $\{0, 1, \ldots, q-1\}$) and is itself output. If $q = 2^m$, one first converts β to an octet string by FE2OS and subsequently converts this octet string to an integer by calling the primitive OS2I.

*** Converting elliptic curve points to octet strings (EC2OS)**

The point $\mathcal{O}$ at infinity (on an elliptic curve over $\mathbb{F}_q$) is defined by an octet string comprising a single zero octet only. So let $P = (h, k)$ be a finite point. The EC2OS primitive produces an octet string $PO = PC \,||\, H \,||\, K$ which is the concatenation of a single octet PC with octet strings H and K representing h and k respectively. The values of PC and K depend on the type of compression used. One has $PC = 0000SUC\hat{K}$, where

$S = 1$ if and only if the SORT compression is used.
$U = 1$ if and only if uncompressed or hybrid form is used.
$C = 1$ if and only if compressed or hybrid form is used.
$\hat{K} = \hat{k}$ if compression is used, it is 0 otherwise.

The first four bits of PC are reserved for (possible) future use and should be set to 0000 for this version of the standard. H is the octet string of length $\lceil \log_{256} q \rceil$ obtained by converting h using FE2OS. If the compressed form is used, K is the empty octet string, whereas if uncompressed or hybrid form is used, we have $K =$ FE2OS$(k, \lceil \log_{256} q \rceil)$. Finally, for the lossy compression we have $PC = 0000\ 0001$, $H =$ FE2OS$(h, \lceil \log_{256} q \rceil)$ and K is empty. Table 6.2 summarizes all these possibilities. Here, $l := \lceil \log_{256} q \rceil$, and p is an odd prime.

Table 6.2 The EC2OS primitive

Representation	PC	H	K	q
uncompressed	$0000\ 0100$	FE2OS(h,l)	FE2OS(k,l)	All
LSB compressed	$0000\ 001\hat{K}$	FE2OS(h,l)	Empty	$p, 2^m$
LSB hybrid	$0000\ 011\hat{K}$	FE2OS(h,l)	FE2OS(k,l)	$p, 2^m$
SORT compressed	$0000\ 101\hat{K}$	FE2OS(h,l)	Empty	$2^m, p^m$
SORT hybrid	$0000\ 111\hat{K}$	FE2OS(h,l)	FE2OS(k,l)	$2^m, p^m$
lossy compression	$0000\ 0001$	FE2OS(h,l)	Empty	All
point at infinity $\mathcal{O}$	$0000\ 0000$	Empty	Empty	All

*** Converting octet strings to elliptic curve points (OS2EC)**

The OS2EC data conversion primitive takes as input an octet string PO, the length $l = \lceil \log_{256} q \rceil$ and the method of compression. If PO contains only one octet and that octet is zero, $\mathcal{O}$ is output. Otherwise, the elliptic curve point $P = (h, k)$ is computed as follows. OS2EC decomposes $PO = PC \,||\, H \,||\, K$, with PC the first octet and with H an octet string of length l. If PC does not match with the method of compression, OS2EC returns error. Otherwise, it uses OS2FE to compute the field element h. If no or hybrid compression is used, the Y-coordinate k is also computed using OS2FE on K. If (h, k) is not a point on the elliptic curve, error is reported. For the LSB or SORT compression, the Y-coordinate $\bar{k}$ is computed using h and $\hat{k}$. If the hybrid scheme is used and $k \neq \bar{k}$, OS2EC halts after reporting error. If all computations are successful till now, the point (h, k) is output.

Note that the checks for (h, k) being on the curve or for the equality $k = \bar{k}$ are optional and may be omitted. For the lossy compression scheme, the Y-coordinate k is not necessarily uniquely determined from the input octet string PO. In that case, any of the two possibilities is output.

*** Converting ring elements to octet strings (RE2OS)**

Ring elements are elements of the convolution polynomial ring $R := \mathbb{Z}[X]/\langle X^n - 1 \rangle$ and can be identified as polynomials with integer coefficients and of degrees $< n$. The element $a(x) = a_0 + a_1 x + \cdots + a_{n-1} x^{n-1} \in R$ (where $x := X + \langle X^n - 1 \rangle \in R$) is represented by the n-tuple of integers $(a_0, a_1, \ldots, a_{n-1})$. The IEEE draft P1363.1 assumes that the coefficients a_i are available modulo a positive integer $\beta \leqslant 256$. But then each a_i is an integer in $\{0, 1, \ldots, \beta - 1\}$ and can be naturally encoded by a single octet. RE2OS, upon receiving $a(x)$ as input, outputs the octet string $a_0 a_1 \cdots a_{n-1}$ of length n.

An example: Let $n = 7$ and $\beta = 128$. The ring element $a(x) = 2 + 11x + 101x^3 + 127x^4 + 71x^5 = (2, 11, 0, 101, 127, 71, 0)$ is converted to the octet string 02 0b 00 65 7f 47 00.

*** Converting octet strings to ring elements (OS2RE)**

Let an octet string $a_0a_1 \dots a_{n-1}$ of length n be given, which we want to convert to an element of $R = \mathbb{Z}[X]/\langle X^n - 1\rangle$. Once again a modulus $\beta \leqslant 256$ is assumed, so that each octet a_i can be viewed as an integer reduced modulo β. Making the natural identification of a_i with an integer, the polynomial $a_0 + a_1x + \cdots + a_{n-1}x^{n-1} \in R$ is output. Thus, for example, the octet string 02 0b 00 65 7f 47 00 gets converted to the ring element $2 + 11x + 101x^3 + 127x^4 + 71x^5$.

*** Converting ring elements to bit strings (RE2BS)**

The RE2BS primitive assumes that the modulus β is a power of 2, that is, $\beta = 2^t$ for some positive integer $t \leqslant 8$. Let a ring element $a(x) = a_0 + a_1x + \cdots + a_{n-1}x^{n-1} \in R$ be given, where each $a_i \in \{0, 1, \dots, \beta - 1\}$. One applies the I2BS primitive on each a_i to generate the bit string $a_{i,0}a_{i,1} \dots a_{i,t-1}$ of length t. The concatenated bit string

$$a_{0,0}a_{0,1} \dots a_{0,t-1}\ a_{1,0}a_{1,1} \dots a_{1,t-1}\ \dots\ a_{n-1,0}a_{n-1,1} \dots a_{n-1,t-1}$$

of length nt is then returned by RE2BS.

As before, take the example of $n = 7$, $\beta = 128 = 2^7$ (so that $t = 7$) and $a(x) = 2 + 11x + 101x^3 + 127x^4 + 71x^5 = (2, 11, 0, 101, 127, 71, 0)$. The coefficients $2, 11, 0, \dots$ should first be converted to bit strings of length 7 each, that is, 2 gives 0000010, 11 gives 0001011 and so on. Thus, the bit string output by RE2BS will be 0000010 0001011 0000000 1100101 1111111 1000111 0000000. Note that here we have shown the bits in groups of 7 in order to highlight the intermediate steps (the outputs from I2BS). With the otherwise standard grouping in blocks of 8, the output bit string looks like 0 00001000 01011000 00001100 10111111 11100011 10000000 and hence transforms to the octet string 00 08 58 0c bf d3 80 by an invocation of BS2OS. This example illustrates that RE2BS followed by BS2OS does not necessarily give the same output as the direct conversion RE2OS, even when every underlying parameter (like β) remains unchanged.

*** Converting bit strings to ring elements (BS2RE)**

Once again we *require* the modulus β to be a power 2^t of 2. Let a bit string $a_0a_1 \dots a_{l-1}$ of length l be given, and we want to compute the ring element $a(x)$ equivalent to this. If l is not an integral multiple of t, the algorithm should quit after reporting error. Otherwise we let $l = nt$ for some $n \in \mathbb{N}_0$, and repeatedly call the BS2I primitive on the bit strings $a_0a_1 \cdots a_{t-1}$, $a_ta_{t+1} \dots a_{2t-1}$, $\dots$, $a_{nt-t}a_{nt-t+1} \dots a_{nt-1}$ to get the integers $\alpha_0, \alpha_1, \dots, \alpha_{n-1}$ respectively. The polynomial $a(x) = \alpha_0 + \alpha_1x + \cdots + \alpha_{n-1}x^{n-1}$ is then output.

We urge the reader to verify that BS2RE with $\beta = 128$ and the bit string

0000010 0001011 0000000 1100101 1111111 1000111 0000000

as input produces the ring element $a(x) = 2 + 11x + 101x^3 + 127x^4 + 71x^5 \in \mathbb{Z}[X]/\langle X^7 - 1\rangle$.

*** Converting binary elements to octet strings (BE2OS)**

A binary (ring) element is an element $a(x) = a_0 + a_1x + \cdots + a_{n-1}x^{n-1} \in R := \mathbb{Z}[X]/\langle X^n - 1\rangle$ with each $a_i \in \{0, 1\}$. One can convert $a(x)$ to an octet string $A_0A_1 \ldots A_{l-1}$ of any desired length l as follows. We denote the bits in the octet A_i as $A_{i,7}A_{i,6} \ldots A_{i,0}$. Here, the index of the bits increases *from right to left*.

First we rewrite the polynomial $a(x)$ as one of degree $8l - 1$, that is, as $a(x) = a_0 + a_1x + \cdots + a_{8l-1}x^{8l-1}$. If $n \leqslant 8l$, this can be done by setting $a_n = a_{n+1} = \cdots = a_{8l-1} = 0$. On the other hand, if $n > 8l$ and one or more of the coefficients $a_{8l}, a_{8l+1}, \ldots, a_{n-1}$ are non-zero (that is, 1), the above rewriting of $a(x)$ cannot be done and BE2OS terminates after reporting failure.

When the above rewriting of $a(x)$ becomes successful, one sets the bits of the output octets as $A_{0,0} := a_0, A_{0,1} := a_1, \ldots, A_{0,7} := a_7, A_{1,0} := a_8, A_{1,1} := a_9, \ldots, A_{1,7} := a_{15}$, $A_{2,0} := a_{16}, A_{2,1} := a_{17}, \ldots, A_{2,7} := a_{23}$, $\ldots$, $A_{l-1,0} := a_{8l-8}, A_{l-1,1} := a_{8l-7}, \ldots, A_{l-1,7} := a_{8l-1}$.

As an example, take $n = 20$ and consider the binary element $a(x) = 1 + x + x^2 + x^{10} + x^{12} \in \mathbb{Z}[X]/\langle X^{20} - 1\rangle$. First let $l = 1$. Rewriting $a(x)$ as a polynomial of degree 7 is not possible, since the coefficients of x^{10} and x^{12} are 1; so BE2OS outputs error in this case. If $l = 2$, then the output octet string will be 00000111 00010100, that is, 07 14. For $l \geqslant 3$, the first two octets will be 07 and 14 as before, whereas the 3rd through l-th octet will be 00.

The BE2OS primitive can be quite effective for reducing storage requirements. For example, the polynomial $a(x)$ of degree 12 of the previous paragraph, viewed as an element of $\mathbb{Z}[X]/\langle X^{200} - 1\rangle$, can be encoded in just two octets. Of course, by specifying $l \geqslant 3$ one may add $l - 2$ trailing zero octets, if one desires. On the other hand, RE2OS requires exactly 200 octets, whereas RE2BS with $\beta = 128$ followed by BS2OS requires exactly $\lceil (200 \times 7)/8 \rceil = 175$ octets for storing the same $a(x)$.

*** Converting octet strings to binary elements (OS2BE)**

Assume that an octet string $A_0A_1 \ldots A_{l-1}$ of length l is given and the equivalent binary element in $R := \mathbb{Z}[X]/\langle X^n - 1\rangle$ is to be determined. As in the case with BE2OS, we index the bits in the octet A_i as $A_i = A_{i,7}A_{i,6} \ldots A_{i,0}$. Now, consider the polynomial $a(x) = a_0 + a_1x + a_2x^2 + \cdots + a_{8l-1}x^{8l-1}$, where $a_{8i+j} = A_{i,j}$. If $n \geqslant 8l$, we set $a_{8l} = a_{8l+1} = \cdots = a_{n-1} = 0$ and output the binary element $a_0 + a_1x + \cdots + a_{n-1}x^{n-1} \in R$. On the other hand, if $n < 8l$ and $a_n = a_{n+1} = \cdots = a_{8l-1} = 0$, then $a_0 + a_1x + \cdots + a_{n-1}x^{n-1} \in R$ equals the polynomial $a(x)$ and is returned. Finally, if $n < 8l$ and if any of the coefficients $a_n, a_{n+1}, \ldots, a_{8l-1}$ is non-zero, then OS2BE returns error.[3]

[3] In this case, it still makes full algebraic sense to treat $a(x)$ as an element of R, though not in the canonical representation.

For example, assume that the octet string $07\ 14$ is given as input to OS2BE. If $n \leqslant 12$, the algorithm outputs error, because the polynomial $a(x)$ in this case has degree 12. For any $n \geqslant 13$, the binary element $1 + x + x^2 + x^{10} + x^{12} \in \mathbb{Z}[X]/\langle X^n - 1\rangle$ is returned.

6.3 RSA Standards

The *public-key cryptography standards* (*PKCS*) [254] refer to a set of standard specifications proposed by the RSA Laboratories. A one-line description of each of these documents is given in Table 6.3. In the rest of this section, we concentrate only on the documents PKCS #1 and #3.

Table 6.3 Public-key cryptography standards from the RSA Laboratories

Document	Description
PKCS #1	RSA encryption and signature
PKCS #2	Merged with PKCS #1
PKCS #3	Diffie–Hellman key exchange
PKCS #4	Merged with PKCS #1
PKCS #5	Password-based cryptography
PKCS #6	Extension of X.509 public-key certificates
PKCS #7	Syntax of cryptographic messages
PKCS #8	Syntax and encryption of private keys
PKCS #9	Attribute types for use in PKCS #6, #7, #8 and #10
PKCS #10	Syntax for certification requests
PKCS #11	Cryptoki, an application programming interface (API)
PKCS #12	Syntax of transferring personal information (private keys, certificates and so on)
PKCS #13	Elliptic curve cryptography (under preparation)
PKCS #15	Syntax for cryptographic token (like integrated circuit card) information

6.3.1 PKCS #1

PKCS #1 describes RSA encryption and RSA signatures. In this section, we summarize Version 2.1 (dated 14 June 2002) of the standard. This version specifies cryptographically stronger encoding procedures compared to the older versions. More specifically, the optimal asymmetric encryption procedure (OAEP [18]) for RSA encryption is incorporated in the Version 2.0 of PKCS #1, whereas the new probabilistic signature scheme (PSS [19]) is introduced in Version 2.1. This latest draft also includes encryption and signature schemes compatible with older versions (1.5 and 2.0). However, adoption of the new algorithms is strongly recommended for enhanced security.

RSA keys

PKCS #1 Version 2.1 introduces the concept of *multi-prime RSA*, in which the RSA modulus n may have more than two prime divisors. For RSA encryption and decryption to work properly, we only need n to be square-free (Exercise 4.1). Using $u > 2$ prime divisors of n increases efficiency and does not degrade the security of the resulting system much, as long as u is not very large. More specifically, if T is the time for RSA private-key operation *without* CRT, then the cost of this operation with CRT is approximately T/u^2 (neglecting the cost of CRT combination).

So an RSA modulus is of the form $n = r_1 r_2 \cdots r_u$ with $u \geqslant 2$ and with pairwise distinct primes $r_1, \ldots, r_u$. For the sake of conformity with the older versions of the standard, the first two primes are given the alternate special names $p := r_1$ and $q := r_2$. PKCS #1 does not mention any specific way of choosing the prime divisors r_i of n, but encourages use of primes that make factorization of n difficult.

An RSA public exponent is an integer e, $3 \leqslant e \leqslant n-1$, with $\gcd(e, \lambda(n)) = 1$, where $\lambda(n) := \operatorname{lcm}(r_1 - 1, r_2 - 1, \ldots, r_u - 1)$. An RSA public key is a pair (n, e) with n and e chosen as above.

The RSA private key corresponding to (n, e) can be stored in one of the two formats. In the first format, one maintains the pair (n, d) with the private exponent d so chosen as to make $ed \equiv 1 \pmod{\lambda(n)}$. In the second format, one stores the five quantities $(p, q, dP, dQ, qInv)$ and, if $u > 2$, the triples (r_i, d_i, t_i) for each $i = 3, \ldots, u$. The meanings of these quantities are as follows:

$$\begin{aligned} p &= r_1 \\ q &= r_2 \\ dP &\equiv e^{-1} \pmod{p-1} \\ dQ &\equiv e^{-1} \pmod{q-1} \\ qInv &\equiv q^{-1} \pmod{p} \\ d_i &\equiv e^{-1} \pmod{r_i - 1} \\ t_i &\equiv (r_1 \cdots r_{i-1})^{-1} \pmod{r_i} \end{aligned}$$

For the sake of consistency, one should store the CRT coefficient $r_1^{-1} \pmod{r_2}$, that is, $p^{-1} \pmod{q}$. In order to ensure compatibility with older versions of PKCS, $q^{-1} \pmod{p}$ is stored instead.

RSA key operations

The RSA public-key operation is used to encrypt a message or to verify a signature. The PKCS draft calls these primitives RSAEP (encryption primitive) and RSAVP1 (verification primitive). It is implemented in a straightforward manner as in Algorithm 6.1.

Algorithm 6.1 **RSA encryption/signature verification primitive**

Input: RSA public key (n, e) and message/signature representative x.
Output: The ciphertext/message representative y.
Steps:
if ($x < 0$) or ($x \geqslant n$) { Return "Error: representative out of range". }
$y := x^e \pmod{n}$.

The RSA decryption or signature-generation primitive is called RSADP or RSASP1 and is given in Algorithm 6.2. The operation depends on the format in which the private key K is stored. The correctness of the primitive is left to the reader as an easy exercise.

Algorithm 6.2 **RSA decryption/signature generation primitive**

Input: RSA private key K and the ciphertext/message representative y.
Output: The message/signature representative x.
Steps:
if ($y < 0$) or ($y \geqslant n$) { Return "Error: representative out of range". }
if (K is stored in the first format) {
  $x := y^d \pmod{n}$.
} else { /* K is stored in the second format */
  $x_1 := y^{dP} \pmod{p}$.
  $x_2 := y^{dQ} \pmod{q}$.
  $h := (x_1 - x_2)qInv \pmod{p}$.
  $x := x_2 + qh$.
  if ($u > 2$) {
    $R := r_1$.
    for $i = 3, \ldots, u$ {
      $x_i := y^{d_i} \pmod{r_i}$.
      $R := R \times r_{i-1}$.
      $h := (x_i - x)t_i \pmod{r_i}$.
      $x := x + Rh$.
    }
  }
}

RSAES–OAEP encryption scheme

The encryption scheme RSAES–OAEP is based on the optimal asymmetric encryption procedure (OAEP) proposed by Bellare and Rogaway [18, 98]. In this procedure, a string of length slightly less than the size of the modulus n is probabilistically *encoded* using a hash function and the encoded message is subsequently encrypted. The probabilistic encoding makes the encryption procedure semantically secure and (provably) provides resistance against chosen-ciphertext attacks. Under this scheme, an adversary can produce a ciphertext, only if she *knows* the corresponding plaintext. Such an encryption scheme is called *plaintext-aware*. Given an *ideal* hash function, Bellare and Rogaway's OAEP is plaintext-aware.

RSAES–OAEP uses a label L which is hashed by a hash function H. One may take L as the empty string. Other possibilities are not specified in the PKCS draft. SHA-1 (or SHA-256 or SHA-384 or SHA-512) is the recommended hash function. The hash values (in hex) of the empty string under these hash functions are given in Table 6.4.

The length of the hash output (in octets) is denoted by $hLen$. For SHA-1, $hLen = 20$. The RSA modulus n is assumed to be of octet length k. The octet length $mLen$ of

Table 6.4 Hash values of the empty string

Function	Hash of the empty string
SHA-1	da39a3ee 5e6b4b0d 3255bfef 95601890 afd80709
SHA-256	e3b0c442 98fc1c14 9afbf4c8 996fb924 27ae41e4 649b934c a495991b 7852b855
SHA-384	38b060a7 51ac9638 4cd9327e b1b1e36a 21fdb711 14be0743 4c0cc7bf 63f6e1da 274edebf e76f65fb d51ad2f1 4898b95b
SHA-512	cf83e135 7eefb8bd f1542850 d66d8007 d620e405 0b5715dc 83f4a921 d36ce9ce 47d0d13c 5d85f2b0 ff8318d2 877eec2f 63b931bd 47417a81 a538327a f927da3e

the input message M must be $\leqslant k-2hLen-2$. RSAES–OAEP uses a mask-generation function designated as MGF (see Algorithm 6.11 for a recommended realization).

Algorithm 6.3 describes the RSA–OAEP encryption scheme which employs the EME–OAEP encoding scheme described in Algorithm 6.4. The use of a random seed makes the encryption probabilistic. We use the notation $||$ to denote string concatenation and $\oplus$ to denote bit-wise XOR.

Algorithm 6.3 **RSA–OAEP encryption scheme**

Input: The recipient's public key (n, e), the message M (an octet string of length $mLen$) and an optional label L whose default value is the empty string.
Output: The ciphertext C of octet length k.
Steps:
/* Check lengths */
if (L is longer than what H can handle) { Return "Error: label too long". }
/* For example, for SHA-1 the input must be of length $\leqslant 2^{61}-1$ octets. */
if ($mLen > k - 2hLen - 2$) { Return "Error: message too long". }

/* Encode M to EM (EME–OAEP encoding scheme) */
$EM := \text{EME-OAEP-encode}(M, L)$. /* Algorithm 6.4 */

/* RSA encryption */
$m := \text{OS2I}(EM)$. /* Convert octet string to integer */
$c := \text{RSAEP}((n, e), m)$. /* RSA encryption primitive */
$C := \text{I2OS}(c, k)$. /* Convert integer back to octet string */

The matching decryption operation is shown in Algorithm 6.5 which calls the EME–OAEP decoding procedure of Algorithm 6.6. The only error message that the decryption and decoding algorithms issue is *decryption error*. This is to ensure that an adversary cannot distinguish between different kinds of errors, because such an ability of the adversary may lead her to guess partial information about the decryption process and thereby mount a chosen-ciphertext attack.

Algorithm 6.4 **RSA–OAEP encoding scheme**

Input: The message M of octet length $mLen$, the label L.
Output: The EME–OAEP encoded message EM.
Steps:
$lHash := H(L)$.
Generate the padding string PS with $k - mLen - 2hLen - 2$ zero octets.
Generate the data block $DB := lHash \,||\, PS \,||\, 01 \,||\, M$.
Let $seed :=$ a random string of length $hLen$ octets.
Generate the data-block mask $dbMask := \mathrm{MGF}(seed, k - hLen - 1)$.
Generate the masked data-block $maskedDB := DB \oplus dbMask$.
Generate mask for seed $seedMask := \mathrm{MGF}(maskedDB, hLen)$.
Generate the masked seed $maskedSeed := seed \oplus seedMask$.
Generate the encoded message $EM := 00 \,||\, maskedSeed \,||\, maskedDB$.

Algorithm 6.5 **RSA–OAEP decryption scheme**

Input: The recipient's private key K, the ciphertext C to be decrypted and an optional label L (the default value of which is the null string).
Output: The decrypted message M.
Steps:
if (the length of L is more than the limitation of H) or (the length of C is not k octets) or ($k < 2hLen + 2$) { Return "Decryption error". }

$c := \mathrm{OS2I}(C)$. /* Convert octet string to integer */
$m := \mathrm{RSADP}(K, c)$. /* RSA decryption primitive */
$EM := \mathrm{I2OS}(m, k)$. /* Convert integer back to octet string */

$M :=$ EME-OAEP-decode(EM, L). /* Algorithm 6.6 */

Algorithm 6.6 **RSA–OAEP decoding scheme**

Input: The encoded message EM and the label L.
Output: The EME–OAEP decoded message M.
Steps:
$lHash := H(L)$.
Write $EM = Y \,||\, maskedSeed \,||\, maskedDB$, where Y is a single octet, $maskedSeed$ is a string of length $hLen$ octets and $maskedDB$ is a string of length $k - hLen - 1$ octets.
$seedMask := \mathrm{MGF}(maskedDB, hLen)$.
$seed := maskedSeed \oplus seedMask$.
$dbMask := \mathrm{MGF}(seed, k - hLen - 1)$.
$DB := maskedDB \oplus dbMask$.
Try to decompose $DB = lHash' \,||\, PS \,||\, 01 \,||\, M$, where $lHash'$ is of length $hLen$ and PS is a (possibly empty) padding string comprising octets 00 only.
if (DB cannot be decomposed as above) or ($lHash' \neq lHash$) or ($Y \neq 00$) { Return "Decryption error". }

RSASSA–PSS signature scheme with appendix

RSASSA–PSS employs the probabilistic signature scheme proposed by Bellare and Rogaway [19]. Under suitable assumptions about the hash function and the mask-generation function, the RSASSA–PSS scheme produces secure signatures which are also *tight* in the sense that forging RSASSA–PSS signatures is computationally equivalent to inverting RSA.

Algorithm 6.7 **RSASSA–PSS signature generation**

Input: The message M (an octet string) to be signed, the private key K of the signer.
Output: The signature S (an octet string of length k).
Steps:
$EM := \text{EMSA–PSS–encode}(M, modBits - 1)$. /* Encode by Algorithm 6.8 */
$m := \text{OS2I}(EM)$. /* Convert octet string to integer */
$s := \text{RSASP1}(m)$. /* RSA signature generation primitive */
$S := \text{I2OS}(s, k)$. /* Convert integer back to octet string */

Algorithm 6.8 **RSASSA–PSS encoding**

Input: The message M to be encoded (an octet string), the maximum bit length $emBits$ of $\text{OS2I}(EM)$. One should have $emBits \geqslant 8hLen + 8sLen + 9$.
Output: The encoded message EM, an octet string of length $emLen := \lceil emBits/8 \rceil$.
Steps:
if (M is longer than what H can handle) { Return "Error: message too long". }
Generate the hashed message $mHash := H(M)$.
if ($emLen < hLen + sLen + 2$) { Return "Encoding error". }
Let $salt :=$ a random string of length $sLen$ octets.
Generate the salted message $M' := 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00 \,||\, mHash \,||\, salt$.
Generate the hashed salted message $mHash' := H(M')$.
Generate the padding string PS with $emLen - sLen - hLen - 2$ zero octets.
Generate the data block $DB := PS \,||\, 01 \,||\, salt$.
Generate the data block mask $dbMask := \text{MGF}(mHash', emLen - hLen - 1)$.
Generate the masted data block $maskedDB := DB \oplus dbMask$.
Set to 0 the leftmost $8emLen - emBits$ bits of the leftmost octet of $maskedDB$.
Compute $EM := maskedDB \,||\, mHash' \,||\, \text{bc}$.

RSASSA–PSS signature generation (Algorithm 6.7) uses the EMSA–PSS encoding method (Algorithm 6.8). Verification (Algorithm 6.9) uses the EMSA–PSS decoding method (Algorithm 6.10). We assume that k is the octet length of the RSA modulus n. Let $modBits$ denote the *bit length* of n. The encoded message is of length $emLen = \lceil (modBits - 1)/8 \rceil$ octets. The probabilistic behaviour of the encoding scheme is incorporated by the use of a random *salt*, the octet length of which is $sLen$. A hash function H that produces hash values of octet length $hLen$ is employed.

Algorithm 6.9 **RSASSA–PSS signature verification**

Input: The message M, the signature S to be verified and the signer's public key (n, e).
Output: Verification status of the signature.
Steps:
if (the length of S is not k octets) { Return "Signature not verified". }
$s := \mathrm{OS2I}(S)$. /* Convert octet string to integer */
$m := \mathrm{RSAVP1}((n, e), s)$. /* RSA signature verification primitive */
$EM := \mathrm{I2OS}(m, emLen)$. /* Convert integer back to octet string */
$status := \mathrm{EMSA\text{–}PSS\text{–}decode}(M, EM, modBits - 1)$. /* Algorithm 6.10 */
if ($status$ is "consistent") { Return "Signature verified". }
else { Return "Signature not verified". }

Algorithm 6.10 **RSASSA–PSS decoding**

Input: The message M (an octet string), the encoded message EM (an octet string of length $emLen = \lceil emBits/8 \rceil$) and the maximum bit length $emBits$ of $\mathrm{OS2I}(EM)$. One should have $emBits \geqslant 8hLen + 8sLen + 9$.
Output: Decoding status: "consistent" or "inconsistent".
Steps:
if (M is longer than what H can handle) { Return "inconsistent". }
Generate the hashed message $mHash := H(M)$.
if ($emLen < hLen + sLen + 2$) { Return "inconsistent". }
Try to decompose $EM = maskedDB \mid\mid mHash' \mid\mid Y$, where
$maskedDB$ is an octet string of length $emLen - hLen - 1$,
$mHash'$ is an octet string of length $hLen$, and Y is a single octet.
if ($Y \neq \mathrm{bc}$) or (the leftmost $8emLen - emBits$ bits of the leftmost octet of $maskedDB$ are not all 0) { Return "inconsistent". }
$dbMask := \mathrm{MGF}(mHash', emLen - hLen - 1)$.
$DB := maskedDB \oplus dbMask$.
Set to 0 the leftmost $8emLen - emBits$ bits of the leftmost octet of DB.
Try to decompose $DB = PS \mid\mid 01 \mid\mid salt$, where PS is a string with
$emLen - sLen - hLen - 2$ zero octets, and $salt$ is of length $sLen$ octets.
if (the above decomposition is unsuccessful) { Return "inconsistent". }
Set $M' := 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00 \mid\mid mHash \mid\mid salt$.
if ($H(M') = mHash'$) { Return "consistent". } else { Return "inconsistent". }

A mask-generation function

A *mask-generation function* (*MGF1*) is specified in the PKCS #1 draft. It is based on a hash function H. The mask-generation function is deterministic in the sense that its output is completely determined by its input. However, the (provable) security of OAEP and PSS schemes are based on the pseudorandom nature of the output of the mask-generation function. This means that any part of the output should be statistically independent of the other parts. MGF1 derives this pseudorandomness from that of the underlying hash function H.

Algorithm 6.11 **Mask-generation function MGF1**

Input: The seed $mgfSeed$ (an octet string) and the desired octet length $maskLen$ of the output mask. One requires $maskLen \leqslant 2^{32}hLen$, where $hLen$ is the octet length of the hash function output.
Output: An octet string $mask$ of length $maskLen$.
Steps:
if ($maskLen > 2^{32}hLen$) { Return "Error: mask too long". }
Initialize T to the empty octet string.
for $i = 0, 1, \ldots, \lceil maskLen/hLen \rceil - 1$ {
  $I := \mathrm{I2OS}(i, 4)$.
  $T := T \mid\mid H(mgfSeed \mid\mid I)$.
}
$mask :=$ the leftmost $maskLen$ octets of T.

The RSA encryption scheme of PKCS #1, Version 1.5

The older encryption scheme RSAES–PKCS1–v1_5 is no longer recommended, since this scheme is *not* plaintext-aware, that is, with high probability, an adversary can generate ciphertexts without knowing the corresponding plaintexts. This allows the adversary to mount chosen-ciphertext attacks. The new drafts of PKCS #1 include this old scheme for backward compatibility. Encryption and decryption for RSAES–PKCS1–v1_5 are given in Algorithms 6.12 and 6.13. Here, k is the octet length of the modulus.

Algorithm 6.12 **RSA–PKCS1 encryption scheme**

Input: The recipient's public key (n, e) and the message M (an octet string).
Output: The ciphertext C which is an octet string of length k.
Steps:
if ($mLen > k - 11$) { Return "Error: message too long". }
Generate a padding string PS of length $k - mLen - 3 \geqslant 8$ octets consisting of random non-zero octets.
Generate the encoded message $EM := 00 \mid\mid 02 \mid\mid PS \mid\mid 00 \mid\mid M$.
$m := \mathrm{OS2I}(EM)$. /* Convert octet string to integer */
$c := \mathrm{RSAEP}((n, e), m)$. /* RSA encryption primitive */
$C := \mathrm{I2OS}(c, k)$. /* Convert integer back to octet string */

Algorithm 6.13 **RSA–PKCS1 decryption scheme**

Input: The recipient's private key K and the ciphertext C (an octet string).
Output: The plaintext message M (an octet string of length $\leqslant k - 11$).
Steps:
if (the length of the ciphertext is not k octets) { Return "decryption error". }
$c := \mathrm{OS2I}(C)$. /* Convert octet string to integer */
$m := \mathrm{RSADP}(K, c)$. /* RSA decryption primitive */
$EM := \mathrm{I2OS}(m, k)$. /* Convert integer back to octet string */
Try to decompose $EM = 00 \mid\mid 02 \mid\mid PS \mid\mid 00 \mid\mid M$, where PS is an octet string of length $\geqslant 8$ and containing only non-zero octets.
if (the above decomposition is unsuccessful) { Return "decryption error". }

The RSA signature scheme of PKCS #1, Version 1.5

The older RSA signature scheme RSASSA–PKCS1–v1_5 is not known to have security loopholes. (Nevertheless, the provably secure PSS scheme is recommended for future applications.) RSASSA–PKCS1–v1_5 uses EMSA–PKCS1–v1_5 message encoding procedure (Algorithm 6.16). The signature generation and verification procedures are given in Algorithms 6.14 and 6.15. Here, k denotes the octet length of the modulus n.

The EMSA–PKCS1–v1_5 message encoding procedure (Algorithm 6.16) uses a hash function H. Although a member of the SHA family is recommended for future applications, MD2 and MD5 are also supported for compliance with older application. An octet string $hashAlgo$ is used whose value depends on the underlying hash algorithm and is given in Table 6.5.

Table 6.5 The string $hashAlgo$ used by EMSA–PKCS1–v1_5

Function	The string $hashAlgo$
MD2	`30 20 30 0c 06 08 2a 86 48 86 f7 0d 02 02 05 00 04 10`
MD5	`30 20 30 0c 06 08 2a 86 48 86 f7 0d 02 05 05 00 04 10`
SHA-1	`30 21 30 09 06 05 2b 0e 03 02 1a 05 00 04 14`
SHA-256	`30 31 30 0d 06 09 60 86 48 01 65 03 04 02 01 05 00 04 20`
SHA-384	`30 41 30 0d 06 09 60 86 48 01 65 03 04 02 02 05 00 04 30`
SHA-512	`30 51 30 0d 06 09 60 86 48 01 65 03 04 02 03 05 00 04 40`

Algorithm 6.14 **RSA–PKCS1 signature generation**

Input: The signer's private key K and the message M to be signed (an octet string).
Output: The signature S (an octet string of length k).
Steps:

Encode M to $EM := \text{EMSA–PKCS1–v1_5}(M, k)$. /* Algorithm 6.16 */
$m := \text{OS2I}(EM)$. /* Convert octet string to integer */
$s := \text{RSASP1}(K, m)$. /* RSA signature generation primitive */
$S := \text{I2OS}(s, k)$. /* Convert integer back to octet string */

Algorithm 6.15 **RSA–PKCS1 signature verification**

Input: The signer's public key (n, e), the message M (an octet string) and the signature S to be verified (an octet string of length k).
Output: Verification status of the signature.
Steps:

if (the length of S is not k octets) { Return "Signature not verified". }
$s := \text{OS2I}(S)$. /* Convert octet string to integer */
$m := \text{RSAVP1}((n, e), s)$. /* RSA signature verification primitive */
$EM' := \text{I2OS}(m, k)$. /* Convert integer back to octet string */
Encode M to $EM := \text{EMSA–PKCS1–v1_5}(M, k)$. /* Algorithm 6.16 */
if ($EM = EM'$) { Return "Signature verified". }
else { Return "Signature not verified". }

Algorithm 6.16 **EMSA–PKCS1 encoding**

Input: The message M (an octet string), the intended length $emLen$ of the encoded message. One requires $emLen \geqslant tLen + 11$, where $tLen$ is the octet length of $hashAlgo$ plus the octet length of the hash output.
Output: The encoded message EM (an octet string of length $emLen$).
Steps:
if (M is longer than what H can handle) { Return "Error: message too long". }
Compute the hash value $mHash := H(M)$.
Let $T := hashAlgo \,||\, mHash$.
/* Let $tLen$ be the octet length of T. */
if ($emLen < tLen + 11$) { Return "Error: encoded message length too short". }
Generate a padding string PS of length $emLen - tLen - 3 \geqslant 8$ octets each having the hexadecimal value ff.
Set $EM := 00 \,||\, 01 \,||\, PS \,||\, 00 \,||\, T$.

6.3.2 PKCS #3

PKCS #3 describes the Diffie–Hellman key-exchange algorithm. The draft assumes the existence of a central authority which generates the domain parameters that include a prime p of octet length k, an integer g satisfying $0 < g < p$ and optionally a positive integer l. The integer g need not be a generator of $\mathbb{F}_p^*$, but is expected to be of sufficiently large multiplicative order modulo p. The integer l denotes the bit length of the private Diffie–Hellman key of an entity. Values of $l \ll 8k$ can be chosen for efficiency. However, for maintaining a desired level of security l should not be too small. Since the central authority determines p, g (and l), individual users need not bother about the generation of these parameters.

During a Diffie–Hellman key-exchange interaction of Alice with Bob, Alice performs the steps described in Algorithm 6.17. Bob performs an identical operation which is omitted here.

Algorithm 6.17 **PKCS3 Diffie–Hellman key-exchange scheme**

Input: p, g and optionally l.
Output: The shared secret SK (an octet string of length k).
Steps:
Alice generates a random $x \in \{1, 2, \ldots, p-2\}$.
/* If l is specified, one should have $2^{l-1} \leqslant x < 2^l$. */
Alice computes $y := g^x \pmod{p}$.
Alice converts y to an octet string $PV := \mathrm{I2OS}(y, k)$.
Alice sends the public value PV to Bob.
Alice receives Bob's public value PV'.
Alice converts PV' to the integer $y' := \mathrm{OS2I}(PV')$.
Alice computes $z := (y')^x \pmod{p}$ (with $0 < z < p$).
Alice transforms z to the shared secret $SK := \mathrm{I2OS}(z, k)$.

Chapter Summary

In this chapter, we describe some standards for representation of cryptographic data in various formats and for conversion of data among different formats. We also present some standard encoding and decoding schemes that are applied before encryption and after decryption. These standards promote easy and unambiguous interfaces with the cryptographic primitives described in the previous chapter.

The *IEEE P1363* range of standards defines several data types: *bit strings*, *octet strings*, *integers*, *prime finite fields*, *finite fields of characteristic* 2, *extension fields of odd characteristics*, *elliptic curves*, *elliptic curve points* and *polynomial rings*. The IEEE drafts also prescribe standard ways of converting data among these formats. For example, the primitive BS2OS converts a bit string to an octet string, the primitive FE2I converts a finite-field element to an integer.

We subsequently mention some of the *public-key cryptography standards* (*PKCS*) propounded by *RSA Laboratories*. Draft *PKCS #1* deals with RSA encryption and signature. In addition to the standard RSA moduli of the form pq, it also suggests possibility of using *multi-prime RSA*, that is, moduli which are products of more than two (distinct) primes. The draft recommends use of the *optimal asymmetric encryption procedure* (*OAEP*). This *probabilistic encryption scheme* provides provable security against chosen-ciphertext attacks. A *probabilistic signature scheme* is also advocated for use. These probabilistic schemes call for using a *mask-generation function* (MGF). A concrete realization of an MGF is also provided. Draft *PKCS #3* standardizes the Diffie–Hellman key exchange algorithm.

Suggestions for Further Reading

The P1363 class of preliminary drafts [134] published by IEEE and the PKC standards [254] from RSA Security Inc. are available for free download from Internet sites. However, IEEE's published standard 1363-2000 is to be purchased against a fee. In addition to the data types and data conversion primitives described in this chapter, the IEEE drafts (P1363, P1363a, P1363.1 and P1363.2) provide encryption/decryption and signature generation/verification primitives and also several encryption and signature schemes based on these primitives. These schemes are very similar to the algorithms that we described in Chapter 5. So we avoided repetition of the same descriptions here. Elaborate encoding procedures are described in the PKCS drafts, but for only RSA- and Diffie–Hellman-based systems. We have reproduced the details in this chapter. The remaining PKCS drafts deal with topics that this book does not directly deal with. A good exception is PKCS #13 that talks about elliptic curve cryptography. This draft is not ready yet; when it is, it may be consulted to learn about the RSA Laboratories' standards on elliptic-curve cryptography.

At present, the different families of standards do not seem to have mutually conflicting specifications. The IEEE has a (free) mailing list for promoting the development and improvement of the IEEE P1363 standards, via e-mail discussions.

Other Internet Standards include the Federal Information Processing Standards or FIPS [221] from NIST, and RFCs (Request for Comments) from the Internet Engineering Task Force or (IETF) [135].

Cryptanalysis in Practice

7

7.1 Introduction
7.2 Side Channel Attacks
7.3 Backdoor Attacks
Chapter Summary
Sugestions for Further Reading

A man cannot be too careful in the choice of his enemies.
— Oscar Wilde (1854–1900), The Picture of Dorian Gray, 1891

If you reveal your secrets to the wind
you should not blame the wind for revealing them to the trees.
— Kahlil Gibran (1883–1931)

There are two ways of constructing a software design:
One way is to make it so simple that there are obviously no deficiencies, and
the other way is to make it so complicated that there are no obvious deficiencies.
The first method is far more difficult.
— Charles Antony Richard Hoare

7.1 Introduction

The security of public-key cryptographic protocols is based on the apparent intractability of solving some computational problems. If one can factor large integers efficiently, one breaks RSA. In that sense, seeking for good algorithms to solve these problems (like factoring integers) is part of cryptanalysis. Proving that no poly-time algorithm can break RSA enhances the status of the security of the protocol from *assumed* to *provable*. On the other hand, developing a poly-time algorithm for breaking RSA (or for factoring integers) makes RSA (and many other protocols) unusable. A temporary set-back to our existing cryptographic tools as it is, it enriches our understanding of the computational problems. In short, breaking the *trapdoors* of public-key cryptosystems is of both theoretical and practical significance.

But research along these mathematical lines is open-ended. A desperate cryptanalyst may not wait indefinitely for a theoretical negotiation. She tries to find loopholes in the systems, that she can effectively exploit to gain secret information.

A cryptographic protocol must be implemented (in software or hardware) before it can be used. Careless implementations often supply the loopholes that cryptanalysts wait for. For example, a software implementation of a public-key system may allow the private key to be read only from a secure device (a removable medium, like CDROM), but may make copies of the key in the memory of the machine where the decryption routine is executed. If the decryption routine does not lock and eventually flush the memory holding the key, a second user having access to the machine can simply read off the secrets.

Software and hardware implementations often tend to leak out secrets at a level much more subtle than the example just mentioned. A public-key algorithm is a *known* algorithm and involves a sequence of well-defined steps dictated by the private key. Each step requires its private share of execution time and power consumption. Watching the decrypting device carefully during a private-key operation may reveal information about the exact sequence of basic steps in the algorithm. Random hardware faults during a private-key operation may also compromise security. Such attacks are commonly dubbed as *side-channel attacks*.

Let us now look at another line of attack. Every user of cryptography is not expected to implement all the routines she uses. On the contrary, most users run precompiled programs available from third parties. How will a user assess the soundness of the products she is using, that is, who will guarantee that there are no (intentional or unintentional) security snags in the products? The key generation software available from a malicious software designer may initiate a clandestine e-mail every time a key pair is generated. It is also possible that a private key supplied by such a program is generated from a small predefined set known to the designer. Even when private keys *look* random, they need not come with the desired unpredictability necessary for cryptographic usage. Such attacks during key generation are called *backdoor attacks*.

In short, public-key cryptanalysis at present encompasses trapdoors, backdoors and side channels. The trapdoor methods have already been discussed in Chapter 4. In this chapter, we concentrate on the other attacks on public-key systems.

7.2 Side-Channel Attacks

Side-channel attacks refer to a class of cryptanalytic tools for determining a private key by measuring *signals* (like timing, power fluctuation, electromagnetic radiation) from or by inducing faults in the device performing operations involving the private key. In this section, we describe three methods of side-channel cryptanalysis: timing attack, power attack and fault attack.

7.2.1 Timing Attack

Paul C. Kocher introduced the concept of side-channel cryptanalysis in his seminal paper [155] on timing attacks. Though not unreasonable, timing attacks are somewhat difficult to mount in practice.

Details of the attack

The private-key operation in many cryptographic systems (like RSA or discrete-log-based systems) is usually a modular exponentiation of the form

$$y := x^d \pmod{n},$$

where d is the private key. The private-key procedure may involve other overheads (like message decoding), but the running time of the routine is usually dominated by and so can be approximated by the time of the modular exponentiation.

Assume that this exponentiation is carried out by a square-and-multiply algorithm known to Carol, the attacker. For example, suppose that Algorithm 3.9 is used. Each iteration of the for loop involves a modular squaring followed conditionally by a modular multiplication. The multiplication is done in an iteration if and only if the corresponding bit e_i in the exponent is 1. Thus, an iteration runs slower if $e_i = 1$ than if $e_i = 0$. If Carol could measure the timing of each individual iteration of the for loop, she would correctly guess most (if not all) of the bits in the exponent. But it is unreasonable to assume that an attacker can collect such detailed timing data. Moreover, if Algorithm 3.10 is used, these detailed data do not help much, because in this case the timing of an individual iteration of the for loop can at best differentiate between the two cases $e_i = 0$ and $e_i \neq 0$. There are $2^t - 1$ non-zero values for each e_i.

However, it is not difficult to think of a situation where the attacker can measure, to a reasonable accuracy, the *total* time of the exponentiation. In order to guess d, Carol requires the times of the modular exponentiations for several different values of x, say $x_1, \ldots, x_k$, all *known* to her. (Note that x_i may be messages to be signed or intercepted ciphertexts.) The same exponent d is used for all these exponentiations. Let T_i be the time for computing $x_i^d \pmod{n}$, as measured by Carol. We may assume that all these k exponentiations are carried out on the same machine using the same routine.

Kocher considers the attack on the exponentiation routine of RSAREF, a cryptography toolkit available from the RSA Laboratories. This routine implements Algorithm 3.10 with $t = 2$. For the sake of convenience, the algorithm is reproduced below. We may assume that the exponent has an even number of bits—if not, pad a leading zero.

Algorithm 7.1 **RSAREF's exponentiation routine**

Input: $n \in \mathbb{N}$, $x \in \mathbb{Z}_n$ and $d = (d_{2l-1}d_{2l-2}\ldots d_1d_0)_2$.
Output: $y := x^d \pmod n$.
Steps:

```
(1)  z_1 := x.
(2)  z_2 := z_1 x (mod n).
(3)  z_3 := z_2 x (mod n).
(4)  y := 1.
(5)  for j = l − 1, ..., 0 {
(6)      y := y^2 (mod n).
(7)      y := y^2 (mod n).
(8)      if ((d_{2j+1} d_{2j})_2 ≠ 0) {
(9)          y := y z_{(d_{2j+1} d_{2j})_2} (mod n).
(10)     }
(11) }
```

Every step of the above algorithm runs in a time dependent on the operands. For example, the modular multiplication in Step (9) takes time dependent on the operands y and $z_{(d_{2j+1}d_{2j})_2}$. The variation in the timing depends on the implementation of the modular arithmetic routines and also on the machine's architecture. However, we make the assumption that for fixed operands each step requires a constant time on a given machine (or on identical machines). This is actually a loss of generality, since the running time of a complex step (like modular multiplication or squaring) for fixed operands may vary for various reasons like process scheduling, availability of cache, page faults and so on. It may be difficult, perhaps impossible, for an attacker to arrange for herself a verbatim emulation of the victim's machine at the time when the latter performed the private-key operations. Let us still proceed with our assumption, say by conceiving of a not-so-unreasonable situation where the effects of these other factors are not sizable enough.

We use the subscript i to denote the i-th private-key operation for $1 \leqslant i \leqslant k$. The entire routine takes time T_i for the i-th exponentiation, that is, for the input x_i. This measurement may involve some (unknown) error which we denote by e_i. The first four steps are executed only once during each call and take a total time of p_i (precomputation time). The for loop is executed l times. We ignore the time needed to maintain the loop (like decrementing j) and also the time taken by the if statement in Step (8). Let $s_{i,j}$ and $t_{i,j}$ be the times taken respectively by Steps (6) and (7), when the loop variable (j) assumes the value j. If Step (9) is executed, we denote by $m_{i,j}$ the time taken by this step, else we set $m_{i,j} := 0$. It follows that

$$T_i = e_i + p_i + \sum_{j=l-1}^{0} \left(s_{i,j} + t_{i,j} + m_{i,j}\right), \tag{7.1}$$

where the index in the sum decreases from $l-1$ to 0 in steps of 1. Carol does not know this break-up (that is, the explicit values of e_i, $s_{i,j}$, $t_{i,j}$ and $m_{i,j}$), but she can make an inductive guess in the following way.

Carol manages a machine and a copy of the exponentiation software both identical to those of the victim. She then successively guesses the secret bit pairs $d_{2l-1}d_{2l-2}$,

$d_{2l-3}d_{2l-4}$, $d_{2l-5}d_{2l-6}$ and so on. Assume that at some stage Carol has correctly determined the exponent bits $d_{2j+1}d_{2j}$ for $j = l-1, l-2, \ldots, j'+1$. Initially $j' = l-1$. Using this information Carol computes $d_{2j'+1}d_{2j'}$ as follows. Carol's knowledge at this stage allows her to measure p_i and $s_{i,j}, t_{i,j}, m_{i,j}$ for $j = l-1, \ldots, j'+1$ — she simply runs Algorithm 7.1 on x_i. Carol then enters the loop with $j = j'$. The squaring operations are unconditional. Carol has the exact operands as the victim for the squaring steps. So Carol also measures $s_{i,j'}$ and $t_{i,j'}$.

The bit pair $d_{2j'+1}d_{2j'}$ (considered as a binary integer) can take any one of the four values $g = 0, 1, 2, 3$. Carol measures the time $\tilde{m}_{i,j',g}$ of Step (9) for each of the four choices of g and adds this time to the time taken by the algorithm so far, in order to obtain:

$$\tilde{T}_{i,g} := p_i + \left(\sum_{j=l-1}^{j'+1} (s_{i,j} + t_{i,j} + m_{i,j}) \right) + (s_{i,j'} + t_{i,j'} + \tilde{m}_{i,j',g}) \,. \qquad \textbf{(7.2)}$$

Kocher observed that the distribution of T_i, $i = 1, \ldots, k$, is statistically *related* to that of $\tilde{T}_{i,j',g}$ only for the correct guess g. In order to see how, we subtract Equation (7.2) from Equation (7.1) to get:

$$T_i - \tilde{T}_{i,g} = e_i + (m_{i,j'} - \tilde{m}_{i,j',g}) + \left(\sum_{j=j'-1}^{0} (s_{i,j} + t_{i,j} + m_{i,j}) \right) . \qquad \textbf{(7.3)}$$

Let us assume that the error term e_i is distributed like a random variable E. Similarly suppose that each multiplication (resp. squaring) has the distribution of a random variable M (resp. S). Taking the variance of Equation (7.3) over the values $i = 1, 2, \ldots, k$ and assuming that the sample size k is so large that the sample variances are very close to the variances of the respective random variables, we obtain:

$$\mathrm{Var}(T_i - \tilde{T}_{i,g}) = \mathrm{Var}(E) + \mathrm{Var}(m_{i,j'} - \tilde{m}_{i,j',g}) + 2j' \,\mathrm{Var}(S) + \lambda \mathrm{Var}(M), \qquad \textbf{(7.4)}$$

where λ denotes the number of times Step (9) is executed for $j = j'-1, \ldots, 0$. Note that λ is dependent on the private key and not on the arguments to the exponentiation routine. For the correct guess g, we have $m_{i,j'} = \tilde{m}_{i,j',g}$ and so

$$\mathrm{Var}(T_i - \tilde{T}_{i,g}) = \mathrm{Var}(E) + 2j' \,\mathrm{Var}(S) + \lambda \mathrm{Var}(M) \,.$$

On the other hand, for an incorrect guess g we have:

$$\mathrm{Var}(T_i - \tilde{T}_{i,g}) = \mathrm{Var}(E) + 2j' \,\mathrm{Var}(S) + (\lambda + 1) \,\mathrm{Var}(M) \,,$$

if one of $m_{i,j'}$ or $\tilde{m}_{i,j',g}$ is zero, or

$$\mathrm{Var}(T_i - \tilde{T}_{i,g}) = \mathrm{Var}(E) + 2j' \,\mathrm{Var}(S) + (\lambda + 2) \,\mathrm{Var}(M) \,,$$

if both $m_{i,j'}$ and $\tilde{m}_{i,j',g}$ are non-zero. (Recall that $\mathrm{Var}(\alpha X + \beta Y) = \alpha^2 \,\mathrm{Var}(X) + \beta^2 \,\mathrm{Var}(Y)$ for any real α, β.)

Calculation of the sample variances of $T_i - \tilde{T}_{i,g}$ for the four choices of g gives Carol a handle to determine (or guess) the correct choice. Carol simply takes the g for which

the variance is minimum. This is the fundamental observation that makes the timing attack work.

Of course, statistical irregularities exist in practice, and the approximation of the actual variances by the sample variances introduces errors in Equation (7.4). These errors are of particular concern for large values of j', that is, during the beginning of the attack. However, if an incorrect guess is made at a certain stage, this is detected soon with high probability, as Carol proceeds further. Suppose that an erroneous guess of $d_{2j''+1}d_{2j''}$ has been made for some $j'' > j'$. This means that the values of y are different from the actual values starting from the iteration of the loop with $j = j'' - 1$. (We may assume that most, if not all, $x_i \neq 1$.) We then do not have a cancellation of the timings for $j = j'' - 1, \ldots, j'$. More correctly, if the guesses for $j = l - 1, \ldots, j'' + 1$ are correct and the first error occurs at $j = j''$, then denoting the subsequent timings by $\hat{s}_{i,j}, \hat{t}_{i,j}, \hat{m}_{i,j}$ one gets

$$\begin{aligned} T_i - \tilde{T}_{i,g} &= e_i + (m_{i,j''} - \hat{m}_{i,j''}) + \\ &\quad \left(\sum_{j=j''-1}^{j'+1} ((s_{i,j} - \hat{s}_{i,j}) + (t_{i,j} - \hat{t}_{i,j}) + (m_{i,j} - \hat{m}_{i,j})) \right) + \\ &\quad (s_{i,j'} - \hat{s}_{i,j'}) + (t_{i,j'} - \hat{t}_{i,j'}) + (m_{i,j'} - \tilde{m}_{i,j',g}) + \\ &\quad \left(\sum_{j=j'-1}^{0} (s_{i,j} + t_{i,j} + m_{i,j}) \right). \end{aligned} \tag{7.5}$$

Since each of the square and multiplication operations takes y as an operand, the original timings and the measured timings (the ones with hat) behave like independent variables and, therefore, taking the variance of Equation (7.5) yields

$$\mathrm{Var}(T_i - \tilde{T}_{i,g}) = \begin{cases} \mathrm{Var}(E) + 2j'' \,\mathrm{Var}(S) + \lambda' \,\mathrm{Var}(M) & \text{if } g = 0 \\ \mathrm{Var}(E) + 2j'' \,\mathrm{Var}(S) + (\lambda' + 1) \,\mathrm{Var}(M) & \text{if } g \neq 0 \end{cases}$$

for some λ' depending on the private key and on the previous guesses, but independent of the current guess g. In other words, Carol loses a meaningful relation of $\mathrm{Var}(T_i - \tilde{T}_{i,g})$ with the correctness of the current guess. Once Carol notices this, she backtracks and changes older guesses until the expected behaviour is restored. Thus, the timing attack comes with an error detection and correction strategy.

An analysis done by Kocher (neglecting E and assuming normal distributions for S and M) shows that Carol needs $k = \mathrm{O}(l)$ for a good probability of success.

Countermeasures

There are several ways in which timing attacks can be prevented.

- If every multiplication step takes exactly the same time and so does every squaring step, the above timing attack does not work. Thus, forcing each multiplication and each squaring take the same respective times independent of their operands disallows Carol to mount the timing attack. Making $m_{i,j}$ constant alone does not suffice, for difference in square timings can be exploited in subsequent iterations

to correct a guess. Forcing every operation take exactly the same time as the slowest possibility makes the implementation run slower. Moreover, finding the slowest possibility may be difficult.

- Interleaving random delays also makes timing attacks difficult to mount, because the attacker then requires more number of samples in order to smooth out the effect of the delays. But again adding delays harms performance and does not completely rule out the possibility of timing attacks.

- Perhaps the best strategy to thwart timing attacks is to use a random pair (u, v) with $v := u^{-d} \pmod{n}$ for each private-key operation. Initially x is multiplied by u and then the product ux is exponentiated to get $u^d x^d \equiv v^{-1} y \pmod{n}$. Multiplication by v then yields the desired y. A new random pair (u, v) must be used for every exponentiation. However, the exponentiation $v := u^{-d} \pmod{n}$ is too costly to be performed during every private-key operation and may itself invite timing attacks. A good trade-off is to choose (u, v) once, keep it secret and for the next private-key operation update (and replace) the old (u, v) by (u', v') with $u' \equiv u^e \pmod{n}$ and $v' \equiv v^e \pmod{n}$ for some small e (random or deterministic). The choice $e = 2$ is quite satisfactory in practice—performing two modular squares is much cheaper than computing the full exponentiation $v := u^{-d} \pmod{n}$.

7.2.2 Power Analysis

In connection with timing attacks, we mentioned that if an adversary were able to measure the timing of each iteration of the square-and-multiply loop during an RSA (or discrete-log-based) private-key exponentiation, she could guess the bits in the key quite efficiently from only a few timing measurements. But it is questionable if such detailed timing data can be made available.

Now, think of a situation where Carol can measure patterns of power consumption made by the decrypting (or signing) device during one or more private-key operations with Alice's private key. If Alice carries out the private-key operations in her personal workstation, it is difficult for Carol to conduct such measurements. So assume that Alice is using a smart card with a device to which Carol has a control. Carol inserts a small resistor in series with the line which drives Alice's smart card. The power consumed by the smart-card circuit is roughly proportional to the current through the resistor. Measuring the voltage across the resistor (and multiplying by a suitable factor) Carol can observe the power consumed by Alice's decryption device. Carol has to use a power measuring device that takes readings at a high frequency (100 MHz to several GHz depending on the budget of Carol). A set of power measurements obtained during a cryptographic operation is called a *power trace*. We now study how power traces can reveal Alice's secrets.

Simple power analysis (SPA)

The individual steps in a private-key operation may be nakedly exposed in a power trace. This is, in particular, the case when different steps consume different amounts

of power and/or take different times. Obtaining information about the operation of the decrypting device and/or the secrets by a direct interpretation of power traces is referred to as *simple power analysis* or *SPA* in short.

As an example of SPA, consider an implementation of RSA exponentiation using the naive square-and-multiply Algorithm 3.9. Here, the most power-consuming operations are modular squaring and modular multiplication. Modular multiplication typically runs slower than modular squaring. Also modular multiplication requires two different operands to fetch from the memory, whereas modular squaring requires only one operand. Thus, a multiplication operation has more and longer power requirements than a squaring operation.

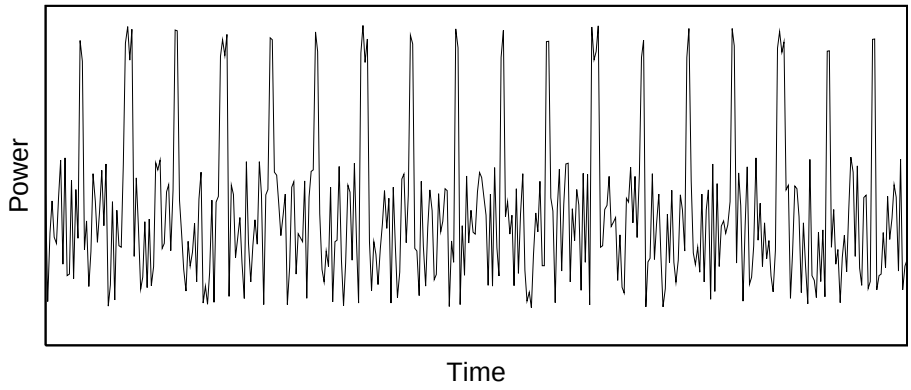


Figure 7.1 Simulated SPA trace for a portion of an RSA private-key operation

A hypothetical[1] SPA trace during a portion of an RSA private-key operation is shown in Figure 7.1. Each spike in the trace corresponds to either a square or a multiplication operation. Let us assume that the power consumption is measured with sufficient resolution, so that no spike is missed. Since multiplication runs longer (and requires more operands) than squaring, multiplication spikes are wider than squaring spikes.

Let us denote a squaring operation by S and a multiplication operation by M. We observe that Alice's smart card performs the sequence

$$SMSMSSMSSSSMSSSMSS$$

of operations during the measurement interval shown. Since multiplication in an iteration of the loop is skipped if and only if the corresponding bit in the exponent is zero, we can group the operations as

$$(SM)(SM)(S)(SM)(S)(S)(S)(SM)(S)(S)(SM)(S)(S\,.$$

This, in turn, reveals the bit string 110100010010 in Alice's private key.

[1] SPA traces from real-life experiments on smart cards, as reported in several references, look similar to this. We, however, generated the trace using a random number generator. Absolute conformity to reality is not always crucial for the purposes of illustration.

Effective as it appears, SPA, in practice, does not pose a huge threat to the security of conventional cryptographic systems. Using algorithms for which power traces do not bear direct relationships with the bits of the private key largely reduces risks of fruitful SPA. The inefficient repeated square-and-multiply Algorithm 7.2 always performs a multiplication after squaring and thereby eliminates chances of a successful SPA.

Algorithm 7.2 **SPA-resistant exponentiation**

Input: $n \in \mathbb{N}$, $x \in \mathbb{Z}_n$ and the private key $d = (d_{l-1} \ldots d_1 d_0)_2$.
Output: $y := x^d \pmod{n}$.
Steps:
$y := 1$.
for $(j = l-1, \ldots, 0)$ {
  $t_0 := y^2 \pmod{n}$.
  $t_1 := t_0 x \pmod{n}$.
  $y := t_{d_j}$.
}

Using the (more efficient) Algorithm 7.1 also frustrates SPA. Some chunks of two successive 0 bits are anyway revealed by power traces collected during the execution of this algorithm. But, for a decently large and random private key, this still leaves Carol with many unknown bits to be guessed. Note, however, that neither of the three remedies suggested to thwart the timing attack on Algorithm 7.1 seems to be effective in the context of SPA. Delays normally do not consume much power (unless some power-intensive dummy computations fill up the delays). Also, the masking of (x, y) by (u, v) fails to produce any alteration in the power consumption pattern during exponentiation.

If some private-key algorithm has unavoidable branchings due to individual bits in the private key, SPA can prove to be a notorious botheration.

Differential power analysis (DPA)

A carefully designed algorithm (like Algorithm 7.2) does not reveal key information from a simple observation of power traces. Moreover, the observed power traces may be corrupted by noise to an extent where SPA is not feasible. In such cases, *differential power analysis* (*DPA*) often helps the cryptanalyst reduce the effects of noise and exploit subtle correlation of power consumption patterns with specific bits in the operands. DPA requires availability of power traces from several private-key operations with the same key.

Consider the SPA-resistant Algorithm 7.2. Suppose that k power traces $P_1(t)$, $\ldots, P_k(t)$ for the computations of $y_i := x_i^d \pmod{n}$, $i = 1, \ldots, k$, are available to Carol, that the ciphertexts $x_1, \ldots, x_k$ are known to Carol and that $d = (d_{l-1} \ldots d_1 d_0)_2$. Carol successively guesses the bits $d_{l-1}, d_{l-2}, d_{l-3}, \ldots$ of the exponent. Suppose that Carol has correctly guessed d_j for $j = l-1, \ldots, j'+1$. She now uses DPA to guess $d_{j'}$.

Let $e := (d_{l-1} d_{l-2} \ldots d_{j'+1})_2$. At the beginning of the for loop with $j = j'$ the variable y holds the value x^e modulo n. The loop computes x^{2e} and x^{2e+1} and assigns

y the appropriate value. If $d_{j'} = 0$, then in the next iteration the loop computes x^{4e} and x^{4e+1}, whereas if $d_{j'} = 1$, then in the next iteration the loop computes x^{4e+2} and x^{4e+3}. It follows that the algorithm handles the value x^{4e} if and only if $d_{j'} = 0$.

For each $i = 1, \ldots, k$, Carol computes $z_i := x_i^{4e} \pmod{n}$. Carol then chooses a particular bit position (say, the least significant bit) and considers the bit b_i of z_i at this position. We make the assumption that there is some subsequent step (or substep) in the implementation for which the average power consumption Π_0 for $b = 0$ is different from the average power consumption Π_1 for $b = 1$.[2]

Carol partitions $\{1, \ldots, k\}$ into two subsets:

$$\begin{aligned} I_0 &:= \{i \mid b_i = 0\}, \\ I_1 &:= \{i \mid b_i = 1\}. \end{aligned}$$

Carol computes the average power traces $\langle P_i(t)\rangle_{i \in I_0}$ and $\langle P_i(t)\rangle_{i \in I_1}$ and subsequently the differential power trace

$$\Delta(t) := \langle P_i(t)\rangle_{i \in I_1} - \langle P_i(t)\rangle_{i \in I_0}.$$

First, let $d_{j'} = 0$. In this case, the routine handles x_i^{4e} and so the power consumption at some time τ is correlated to the bit b_i of x_i^{4e}. At any other instant, the power consumption is uncorrelated to this particular bit value. Therefore, if the sample size is sufficiently large and if the measurement noise has mean at zero, we have:

$$\Delta(t) \approx \begin{cases} \Pi_1 - \Pi_0 & \text{if } t = \tau, \\ 0 & \text{otherwise.} \end{cases}$$

On the other hand, if $d_{j'} = 1$, the value x_i^{4e} never appears in the execution of the algorithm and so at every time t the power consumption is uncorrelated to the particular bit of x_i^{4e} and so we expect

$$\Delta(t) \approx 0 \quad \text{for all } t.$$

Figure 7.2 illustrates the two cases.[3] If the differential power trace has a distinct spike, the guess $d_{j'} = 0$ is correct. So by observing the existence or otherwise of a spike, Carol determines whether $d_{j'} = 0$ or $d_{j'} = 1$.

The number k of samples required for a good probability of success depends on the bias $\Pi_1 - \Pi_0$ relative to the measurement noise. We assume that $\#I_1 \approx \frac{k}{2} \approx \#I_0$. If the noise has a variance of σ^2, then by the central limit theorem the noise in each average power trace $\langle P_i(t)\rangle_{i \in I_1}$ or $\langle P_i(t)\rangle_{i \in I_0}$ has at each t an approximate variance $2\sigma^2/k$, and so in the differential power trace $\Delta(t)$ the noise has an approximate variance $4\sigma^2/k$. In order that the bias $\Pi_1 - \Pi_0$ stands out against the noise, we require $\Pi_1 - \Pi_0 \gg 2\sigma/\sqrt{k}$, say, $\Pi_1 - \Pi_0 \geqslant 8\sigma/\sqrt{k}$, that is, $k \geqslant 64\sigma^2/(\Pi_1 - \Pi_0)^2$.

[2] The exact step which exhibits differential bias toward an individual bit value is dependent on the implementation. If the implementation does not provide such a step, the attack cannot be mounted in this way. Initially, the DPA was proposed for DES, a symmetric encryption algorithm, in which such a dependence is clearly available. With asymmetric-key encryption, such a strong dependence of the power, consumed by a step, on an individual bit value is not obvious. One may, however, use other dividing criteria, like *low versus high Hamming weight* (that is, number of one-bits) in the operand, which bear more direct relationships with power consumption.

[3] Once again, these are hypothetical traces obtained by random number generators.

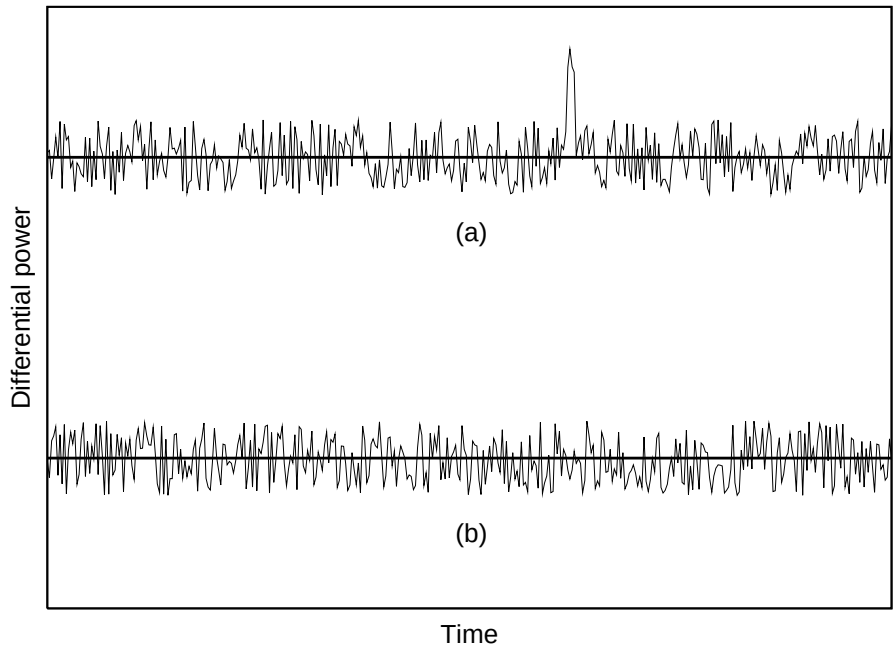


Figure 7.2 Simulated DPA trace for a portion of an RSA private-key operation (a) for the correct guess (b) for an incorrect guess

Countermeasures

Several countermeasures can be adopted to prevent DPA, both in the software level and in the hardware level.

- Interleaving random delays between instructions destroys the alignment of the time τ in different power traces. Using a clock with randomly varying tick-rate has a similar effect. The delays should be such that they cannot be easily analyzed and subsequently removed. Random delays increase the number of samples required for a successful DPA to an infeasible value.
- Suitable implementations of the power-critical steps destroy the power consumption signature of these steps. For example, one may go for an implementation that exhibits a constant power consumption pattern irrespective of the operands. Another possibility is replacement of complex critical instructions by atomic instructions (like assembly instructions) for which the dependence of power consumption on operands is less or difficult to analyze. However, the assumption that one can measure power at any resolution (perhaps at infinite resolution, say, using an analog device) indicates that this countermeasure challenges only the attacker's budget.
- Masking (x, y) by multiplying with (u, v) (as we did to prevent timing attacks) also eliminates chances of mounting successful DPA. One has to use a fresh mask

for each private-key operation. Random unknown masks destroy the correlation of the bit values b_i with power consumption. That is, the chosen bit b_i of x_i^{4e} behaves randomly in relation to the same bit of $(u_i x_i)^{4e}$ and so the differential power trace no longer leaks the bias $\Pi_1 - \Pi_0$.

- Another strategy to foil DPA is to use randomization in the private exponent d. Instead of computing $y := x^d \pmod n$, one chooses a small random integer r (typically of bit size $\leqslant 20$) and computes $y := x^{d+rh} \pmod n$, where h is $\phi(n)$ for RSA or the order of the discrete-log (sub)group. Since $d = O(h)$ typically, the performance of the exponentiation routine does not deteriorate much. But random values of r during different private-key operations change the exponent bits in an unpredictable manner.

- Quick changes in the exponent (the private key, that is, the key pair) also prevent the attacker to gather sufficiently many power traces for mounting a successful DPA. A key-use counter can be employed for this purpose. Whenever a given private key has been used on a small predetermined number of occasions, the key pair is updated.

- Hardware shielding of the decrypting device also reduces DPA possibilities. For example, in-chip buffers between the external power source and the chip processor have been proposed to mask off the variation of internal power from external measurements. Such hardware countermeasures are, in general, somewhat costlier than software countermeasures.

Paul Kocher asserts: *DPA highlights the need for people who design algorithms, protocols, software, and hardware to work closely together when producing security products.*

7.2.3 Fault Analysis

We finally come to the third genre of side-channel cryptanalysis. We investigate how hardware faults occurring during private-key operations can reveal the secret to an adversary. There are situations where a *single* fault suffices. Boneh et al. [30] classify hardware faults into three broad categories.

(1) **Transient faults** These are faults caused by random (unpredictable) hardware malfunctioning. These may be the outcomes of occasional flips of bit values in registers or of temporary erroneous outputs from logic or arithmetic circuits in the processor. These faults are called transient, because they are not repeated. It is rather difficult to detect such (silent) faults.

(2) **Latent faults** These are faults generated by some permanent malfunctioning and/or bugs inherent in the processor. For example, the floating-point bug in the early releases of the Pentium processor may lead to latent faults. Latent faults are permanent, that is, repeated, but may be difficult to locate in practice.

(3) **Induced faults** An induced fault is deliberately caused by an adversary. For example, a short surge of electromagnetic radiation may cause a smart card

to malfunction temporarily. A malicious adversary can induce such temporary hardware faults to extract secret information from the smart card. It is, however, difficult to induce deliberate faults in a remote workstation.

Although induced faults appear to be the ones to guard against most seriously, the other two types of faults are also of relevance. Consider a certifying authority signing many messages. Transient and/or unknown latent faults may reveal the authority's private key to a user who can later utilize this knowledge to produce false certificates.

Fault attack on RSA based on CRT

Consider the implementation of RSA private-key operation based on the CRT combination of the values obtained by exponentiation modulo the prime divisors p and q of the modulus n (Algorithm 5.4). Suppose that m is a message to be signed and $s := m^d \pmod n$ the corresponding signature, where d is the signer's private key. The CRT-based implementation computes $s_1 := s \pmod p$ and $s_2 := s \pmod q$. Assume that due to hardware fault(s) exactly one of s_1 and s_2 is wrongly computed. Say, s_1 is incorrectly computed as $\tilde{s}_1$. The corresponding faulty signature is denoted by $\tilde{s}$. We assume that the CRT combination of $\tilde{s}_1$ and s_2 is correctly computed.

An adversary requires the faulty signature $\tilde{s}$ and the correct signature s on the *same* message m in order to obtain the factor q of n. To see how, note that $\tilde{s} \equiv \tilde{s}_1 \pmod p$, $s \equiv s_1 \pmod p$ and $\tilde{s}_1 \not\equiv s_1 \pmod p$, so that $\tilde{s} \not\equiv s \pmod p$, that is, $p \not| (\tilde{s} - s)$. On the other hand, $\tilde{s} \equiv s_2 \equiv s \pmod q$, that is, $q | (\tilde{s} - s)$. Therefore,

$$q = \gcd(\tilde{s} - s, n)\,.$$

This is how the fault analysis of Boneh et al. [30] works.

Arjen K. Lenstra et al. [142] point out that the knowledge of the faulty signature $\tilde{s}$ alone reveals the secret divisor q, that is, one does not require the genuine signature s on m. The verification key e of the signer is publicly known. Since RSA exponentiation is bijective, $m \not\equiv \tilde{s}^e \pmod n$. However, $m \equiv \tilde{s}^e \pmod q$, and so $m \not\equiv \tilde{s}^e \pmod p$. It follows that

$$q = \gcd(\tilde{s}^e - m, n)\,.$$

Fault attack on RSA without CRT

Now, consider an implementation of RSA decryption based on a single exponentiation modulo n. For such an implementation, several models of fault attacks have been proposed. These attacks are less practical than the attack on CRT-based RSA just mentioned, because now one requires *several* faulty signatures in order to deduce the entire private key. Here, we present an attack due to Bao et al. [17].

As usual, the RSA modulus is $n = pq$ and the signer's key pair is (e, d). Consider a valid signature s on a message m. Let $d = (d_{l-1} \ldots d_1 d_0)_2$ be the binary representation of the private key. Consider the powers:

$$s_i \equiv m^{2^i} \pmod n \quad \text{for } i = 0, 1, \ldots, l-1.$$

The signature s can be written as:

$$s \equiv \prod_{i=0}^{l-1} s_i^{d_i} \pmod{n}.$$

We assume that the attacker knows m and s and hence can compute s_i and s_i^{-1} modulo n for $i = 0, \ldots, l-1$. There is no harm in assuming that the message m is randomly chosen. (We may assume that randomly chosen integers are invertible modulo n, because encountering a non-invertible non-zero integer by chance is a stroke of unimaginable good luck and is tantamount to knowing the factors of n.)

In order to guess a bit of d, the attacker induces a fault in exactly one of the bits d_j, changing it from d_j to $\bar{d}_j = 1 - d_j$. The position j is random, that is, not under the control of the attacker. Now, the algorithm outputs the faulty signature

$$\tilde{s} \equiv s_{l-1}^{d_{l-1}} \cdots s_{j+1}^{d_{j+1}} s_j^{\bar{d}_j} s_{j-1}^{d_{j-1}} \cdots s_0^{d_0} \pmod{n},$$

and so

$$\tilde{s}/s \equiv s_j^{\bar{d}_j}/s_j^{d_j} \equiv \begin{cases} s_j \pmod{n} & \text{if } d_j = 0, \\ s_j^{-1} \pmod{n} & \text{if } d_j = 1. \end{cases}$$

A repetition in the values $s_{l-1}, \ldots, s_0, s_{l-1}^{-1}, \ldots, s_0^{-1}$ modulo n is again an incident of minuscule probability. Hence the attacker can uniquely identify the bit position j and the bit value d_j in d by comparing $\tilde{s}/s$ with these $2l$ values.

Statistical analysis implies that the attacker needs to repeat this procedure about $l \log l$ times (on same or different (m, s) pairs) in order to ensure that the probability of identifying *all* the bits of d is at least $1/2$.

Fault attack on the Rabin digital signature algorithm

Recall from Algorithm 5.34 that the Rabin signature algorithm uses CRT to combine $s_1 \pmod{p}$ and $s_2 \pmod{q}$. Thus, the attack on CRT-based RSA, described earlier, is applicable mutatis mutandis to the Rabin signature scheme. The computation of the square roots s_1 and s_2 demands the major portion of the running time of the routine. Inducing a fault during the execution is, therefore, expected to affect exactly one of s_1 and s_2, as desired by the attacker.

Fault attack on DSA

Bao et al. [17] propose a fault attack on the digital signature algorithm (DSA). We work with the notations of Algorithm 5.43 and Algorithm 5.44, except that, for maintaining uniformity in this section, we use m (instead of M) to denote the message to be signed. The (public) parameters are p, a prime divisor r of $p-1$ of length 160 bits and an element $g \in \mathbb{F}_p^*$ of multiplicative order r. The signer's DSA key pair is $(d, g^d \pmod{p})$ with $1 < d < r$.

Suppose that during the generation of a DSA signature, an attacker induces a fault in exactly one bit position of d changing it to $\tilde{d} := d \pm 2^j$. The routine generates the faulty signature $(s, \tilde{t})$, where

$$\begin{aligned} s &\equiv \left(g^{d'} \pmod p\right) \pmod r, \\ \tilde{t} &\equiv d'^{-1}(H(m) + \tilde{d}s) \pmod r, \end{aligned}$$

$(d', g^{d'})$ being the session key pair (not mutilated). As in the DSA signature-verification scheme, the attacker computes the following:

$$\begin{aligned} \tilde{w} &\equiv \tilde{t}^{-1} \pmod r, \\ \tilde{w}_1 &\equiv H(m)\tilde{w} \pmod r, \\ \tilde{w}_2 &\equiv s\tilde{w} \pmod r, \\ \tilde{S} &\equiv \left(g^{\tilde{w}_1}(g^d)^{\tilde{w}_2} \pmod p\right) \pmod r \\ &\equiv \left(g^{\tilde{w}(H(m)+sd) \pmod r} \pmod p\right) \pmod r. \end{aligned}$$

For each $i = 0, \ldots, l-1$ (where the bit length of d is l), the attacker also computes

$$T_i \equiv \left(g^{\tilde{w}s2^i \pmod r} \pmod p\right) \pmod r.$$

Assume that the j-th bit d_j of d is altered. If $d_j = 0$, $\tilde{d} = d + 2^j$ and so

$$\begin{aligned} \tilde{S}T_j &\equiv \left(g^{\tilde{w}(H(m)+s(d+2^j)) \pmod r} \pmod p\right) \pmod r \\ &\equiv \left(g^{\tilde{w}(H(m)+s\tilde{d}) \pmod r} \pmod p\right) \pmod r \\ &\equiv \left(g^{d'} \pmod p\right) \pmod r \\ &\equiv (s \pmod p) \pmod r. \end{aligned}$$

On the other hand, if $d_j = 1$, then $\tilde{d} = d - 2^j$ and a similar calculation shows that

$$\tilde{S}/T_j \equiv (s \pmod p) \pmod r.$$

Thus, the attacker computes $\tilde{S}T_j$ and $\tilde{S}/T_j$ for all $j = 0, \ldots, l-1$ and notices a unique match (with s). This discloses the position j and the corresponding bit d_j.

Fault attack on the ElGamal signature scheme

A fault attack similar to that on the DSA scheme can be mounted on the ElGamal signature scheme. We here propose an alternative method proposed by Zheng and Matsumoto [315]. The novelty in their approach is that it performs the cryptanalysis of the ElGamal signature scheme by inducing fault on the *pseudorandom bit generator* of the signer's smart card.

Algorithms 5.36 and 5.37 describe the ElGamal signature scheme on a general cyclic group G. Here, we restrict our attention to the specific group $\mathbb{F}_p^*$ (though the

following exposition works perfectly well for a general G). The parameters are a prime modulus p and a generator g of $\mathbb{F}_p^*$. The signer's key-pair is $(d, g^d \pmod p)$ for some d, $2 \leqslant d \leqslant p-2$.

In order to generate a signature (s,t) on a message m, a random session key d' is generated and subsequently the following computations are carried out:

$$\begin{aligned} s &\equiv g^{d'} \pmod p, \\ t &\equiv d'^{-1}(H(m) - dH(s)) \pmod{p-1}. \end{aligned}$$

Zheng and Matsumoto attack the generation of the session key d'. They propose the possibility that an abnormal physical stress (like low voltage) forces a constant output d_0 for d' from the pseudorandom bit generator (software or hardware) in the smart card. First, assume that this particular value d_0 is known a priori to the attacker. She then lets a message m generate a signature (s,t) with the session secret d_0. The private key d is then immediately available from the equation:

$$d \equiv H(s)^{-1}(H(m) - d_0 t) \pmod{p-1}.$$

Here, we assume that $H(s)$ is invertible modulo $p-1$.

If d_0 is not known a priori, the attacker generates two signatures (s_1,t_1) and (s_2,t_2) on messages m_1 and m_2 respectively. Since d' is always d_0, we have $s_1 = s_2 = s_0$, say. One can then easily calculate

$$d_0 \equiv (t_1 - t_2)^{-1}(H(m_1) - H(m_2)) \pmod{p-1},$$

which, in turn, yields

$$d \equiv H(s_0)^{-1}(H(m_1) - d_0 t_1) \pmod{p-1}.$$

Fault attack on the Feige–Fiat–Shamir identification protocol

Let us conclude our repertoire of fault attack examples by explaining an attack on the FFS zero-knowledge identification protocol. This attack is again from Boneh et al. [30].

We use the notations of Algorithm 5.69. A modulus $n = pq$, $p, q \in \mathbb{P}$, is first chosen (by Alice or by a trusted third party). Alice selects random $x_1, \ldots, x_t \in \mathbb{Z}_n^*$ and random bits $\delta_1, \ldots, \delta_t$, computes $y_i := (-1)^{\delta_i}(x_i^2)^{-1} \pmod n$, publishes $(y_1, \ldots, y_t)$ and keeps $(x_1, \ldots, x_t)$ secret.

During an identification session with Bob, Alice generates a random commitment $c \in \mathbb{Z}_n^*$ and sends to Bob the witness $w := c^2 \pmod n$. (For simplicity, we take γ of Algorithm 5.69 to be 0.) When Alice is waiting for a challenge from Bob, a fault occurs in her smart card changing the commitment c to $c + E$. Assume that the fault is at exactly one bit position, that is, $E = \pm 2^j$ for some $j \in \{0, 1, \ldots, l-1\}$, l being the bit length of c (or of n). This fault may be purposely induced by Bob with the malicious intention of guessing Alice's secret $(x_1, \ldots, x_t)$.

Bob then generates a random challenge $(\epsilon_1, \ldots, \epsilon_t) \in \{0,1\}^t$ as usual. Upon reception of this challenge Alice computes and sends to Bob the faulty response

$$\tilde{r} \equiv (c + E) \prod_{i=1}^{t} x_i^{\epsilon_i} \pmod n.$$

The knowledge of $\tilde{r}$ now aids Bob to obtain the product $\prod_{i=1}^{t} x_i^{\epsilon_i} = \prod_{\epsilon_i=1} x_i$ as follows. First, note that

$$\tilde{r}^2 \equiv (c+E)^2 \prod_{i=1}^{t} (x_i^2)^{\epsilon_i} \pmod{n},$$

so that

$$(c+E)^2 \equiv (-1)^{\delta} \tilde{r}^2 \prod_{i=1}^{t} y_i^{\epsilon_i} \pmod{n} \text{ for some } \delta \in \{0,1\}.$$

There are only $4l$ possible values of (E,δ). Bob tries all these possibilities one by one. To simplify matters we assume that only one value of (E,δ) with E of the special form $\pm 2^j$ and with $\delta \in \{0,1\}$ satisfies the last congruence. In practice, the existence of two (or more) solutions for (E,δ) is an extremely improbable phenomenon. For a guess of (E,δ), the commitment c can be computed as

$$\begin{aligned} c &\equiv (2E)^{-1} \left((c+E)^2 - c^2 - E^2\right) \\ &\equiv (2E)^{-1} \left[(-1)^{\delta} \tilde{r}^2 \left(\prod_{i=1}^{t} y_i^{\epsilon_i}\right) - w - E^2\right] \pmod{n}. \end{aligned}$$

The correctness of the guess (E,δ) can be verified from the relation $w \equiv c^2 \pmod{n}$. Bob can now compute the desired product

$$T := \prod_{i=1}^{t} x_i^{\epsilon_i} \equiv (c+E)^{-1} \tilde{r} \pmod{n}.$$

In order to strengthen the confidence about the correctness of T, Bob may repeat the protocol once more with the same values of $\epsilon_1, \ldots, \epsilon_t$, but under normal conditions (that is, without faults). This time he obtains $w' \equiv (c')^2 \pmod{n}$ and $r' \equiv c'T \pmod{n}$, which together give $(r')^2 \equiv w'T^2 \pmod{n}$, a relation that proves the correctness of T.

Bob repeats the above procedure t times in order to generate the system:

$$\begin{aligned} x_1^{\epsilon_{11}} x_2^{\epsilon_{12}} \cdots x_t^{\epsilon_{1t}} &\equiv T_1 \pmod{n} \\ x_1^{\epsilon_{21}} x_2^{\epsilon_{22}} \cdots x_t^{\epsilon_{2t}} &\equiv T_2 \pmod{n} \\ &\vdots \\ x_1^{\epsilon_{t1}} x_2^{\epsilon_{t2}} \cdots x_t^{\epsilon_{tt}} &\equiv T_t \pmod{n} \end{aligned} \tag{7.6}$$

Here, ϵ_{ki} and T_k are known to Bob. Moreover, the exponents ϵ_{ki} can be so selected that the matrix (ϵ_{ki}) is invertible modulo 2. In order to determine x_1, Bob tries to find $(u_1, \ldots, u_t) \in \{0,1\}^t$ satisfying

$$\begin{aligned} T_1^{u_1} T_2^{u_2} \cdots T_t^{u_t} &\equiv \prod_{k=1}^{t} (x_1^{\epsilon_{k1}} x_2^{\epsilon_{k2}} \cdots x_t^{\epsilon_{kt}})^{u_k} \\ &\equiv x_1^{2v_1+1} x_2^{2v_2} \cdots x_t^{2v_t} \pmod{n} \end{aligned}$$

for some integers $v_1, \ldots, v_t$. Comparing the coefficients gives the linear system

$$\sum_{k=1}^{t} u_k \begin{pmatrix} \epsilon_{k1} \\ \epsilon_{k2} \\ \vdots \\ \epsilon_{kt} \end{pmatrix} \equiv \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \pmod 2 ,$$

which can be solved for $u_1, \ldots, u_t$, since the matrix (ϵ_{ki}) is invertible modulo 2. The solution gives $v_1, \ldots, v_t$ and hence

$$x_1 \equiv \pm (y_1^{v_1} y_2^{v_2} \cdots y_t^{v_t}) (T_1^{u_1} T_2^{u_2} \cdots T_t^{u_t}) \pmod n .$$

Similarly, $x_2, \ldots, x_t$ can be determined up to sign. Plugging in these values of x_i in System (7.6) and solving another linear system modulo 2 gives the exact signs of all x_i.

Notice that Bob could have selected $\epsilon_{ki} = \delta_{ki}$ (where δ is the Dirac delta). For this choice, System (7.6) immediately gives $x_1, \ldots, x_t$. But, in practice, Alice may disagree to respond to such simplistic challenges. Moreover, Bob must not raise any suspicion about a possible malpractice. For a general choice, all Bob has to do additionally is a little amount of simple linear algebra. The parameter t is rather small (typically less than 20); so this extra effort is of little concern to Bob.

Countermeasures

Fault analysis could be a serious threat, especially to smart-card users and certification authorities. We mention here some precautions to guard against such attacks. Some of these work for a general kind of fault attack, the others are specific to the algorithms they plan to protect.

- One obvious general strategy is to perform the private-key operation twice and compare the results from the two executions. If the two results disagree, a fault must have taken place. It is then necessary to restart the computation from the beginning. This strategy slows down the implementation by a factor of two. Moreover, latent (permanent) faults cannot be detected by this method—the same error creeps in during every run.

- It is sometimes easier to verify the correctness of the output by performing the reverse operation. For instance, after an RSA signature $s \equiv m^d \pmod n$ is generated, one can check whether $m \equiv s^e \pmod n$. If so, one can be reasonably confident about the correctness of s. If the RSA encryption exponent is small (like 3 or 257), this verification is quite efficient.

- Ad hoc algorithm-specific tricks often offer effective and efficient checks for errors. Shamir [268] proposes the following check for CRT-based RSA signature generation. One chooses a small random prime r (say, of length ~ 32 bits) and computes $s_1 \equiv m^d \pmod{pr}$ and $s_2 \equiv m^d \pmod{qr}$. If $s_1 \not\equiv s_2 \pmod r$, then one or both of the exponentiations went wrongly. If, on the other hand, $s_1 \equiv s_2 \pmod r$, then $s_1 \pmod p$ and $s_2 \pmod q$ are combined by the CRT.

- Maintaining extraneous error-checking data can guard against random bit flips. Parity check bits can detect the existence of single bit flips. Retaining a verbatim copy of a secret information d and comparison of the two copies at strategic instants can help detect more serious faults. It appears unlikely that both the copies can be affected by faults in exactly the same way. For discrete-log-based systems, maintaining d^{-1} in tandem with d appears to be a sound approach. Since the bits of d^{-1} are not in direct relationship with those of d, an attack on d cannot easily produce the relevant changes in d^{-1}. As an example, consider the attack on DSA effected by toggling a bit of the secret key d. The second part of the signature can be generated in two ways: by computing $t_1 \equiv d'^{-1}(H(m) + ds) \pmod r$ using d, and by computing $t_2 \equiv d'^{-1}(d^{-1})^{-1}(d^{-1}H(m) + s) \pmod r$ using d^{-1}. If $t_1 \equiv t_2 \pmod r$, we can be pretty confident that this common value is the correct signature.
- Appending random strings to the messages being signed also prevents timing attacks. Such random strings are not known to the adversary and cannot be easily recovered by the verification routine on a faulty signature. Also in this case the signer signs different strings on different occasions, even when the message remains the same.
- Hardware countermeasures can also be adopted. Adequately shielded cards resist induced faults. In a situation described by Zheng and Matsumoto, the card should refuse to work instead of generating constant *random* bits. In the scenario of fault analysis, it, however, appears that robustness can be implanted easily at the software level. At any rate, sloppy hardware designs are never advocated.

EXERCISE SET 7.2

7.1 Consider the notations of Section 7.2.1. Assume that $m_{i,j}$ is constant for all i, j (and irrespective of $d_{2j+1}d_{2j}$), but the square times $s_{i,j}$ and $t_{i,j}$ vary according to their operands. Device a timing attack on such a system.

7.2 Show that under reasonable assumptions the SPA-resistant Algorithm 7.2 can be cryptanalyzed by timing attacks.

7.3 Recall that SPA of Algorithm 7.1 may leak partial information on the private key (some 00 sequences in the key). Rewrite the algorithm to prevent this leakage.

7.4 Assume that in Bao et al.'s attack on RSA described in the text, the attacker can induce faults in exactly *two* bit positions of d. Suggest how the two bits of d at these positions can be revealed from the resulting faulty signature.

7.5 Consider a variant of the Bao et al.'s attack on RSA described in the text, in which the valid signature s on m is *unknown* to the attacker. Explain how the position j of the erroneous bit and the bit d_j at this position can still be identified. **[H]**

7.6 Bao et al. [17] propose an alternate fault analysis on RSA with square-and-multiply exponentiation. Use the notations (n, e, d, m, s, s_i) as in the text. Assume that the attacker knows an (m, s) pair and can induce a fault in exactly one of the values s_j (and nowhere else) and generate the corresponding faulty signature. Suggest a strategy how the position j and the bit d_j can be recovered in this case.

7.7 Propose a fault attack on the ElGamal signature scheme (Algorithms 5.36 and 5.37), similar to the attack on DSA described in the text.

7.3 Backdoor Attacks

Backdoor attacks on a public-key cryptosystem refer to attacks embedded in the key generation procedure (hardware or software) by the designer of the procedure. A *contaminated cryptosystem* is one in which the key generation procedure comes with hidden backdoors. A good backdoor attack should meet the following criteria:

- To a user, keys generated by the contaminated system should be indistinguishable from those generated by an honest version of the cryptosystem. For example, the parameters and keys must *look* sufficiently random.
- Keys generated by the contaminated system should satisfy the input/output requirements of an honest system. For example, for the RSA cryptosystem the user should be allowed to opt for small public exponents.
- A contaminated key generation procedure should not run (on an average) much slower than the honest procedure.
- The designer (and nobody else) should have the exclusive capability of determining the secret information from a contaminated published public key.
- A user (other than the designer), detecting or suspecting information leakage from a contaminated system, may reverse-engineer the binaries or the smart card to identify the contaminated key generation procedure. The user may even be given the source code of the contaminated routine. Still the user should not be able to steal keys from other users of the same contaminated system. In this sense, a good backdoor protects the designer universally.
- A stronger requirement is that reverse-engineering (or source code) should also not allow a user to distinguish (in poly-time) between keys generated by the contaminated procedure and those generated by a genuine procedure. It is exclusively the designer who should possess the capability to make such distinctions in poly-time.

Young and Yung [307] have proposed using public-key cryptography itself for generating backdoors. In their schemes, the attacker (the designer) embeds the encryption routine and the encryption key of the attacker in the key generation procedure of the contaminated system. The decryption key of the attacker is not embedded in the contaminated system and is known only to the attacker. The attacker's encryption

system is assumed to be honest and unbreakable and, thereby, it gives the attacker the exclusive power to decrypt contaminated keys. Young and Yung call such a backdoor a *secretly embedded trapdoor with universal protection* (*SETUP*). They also coined the term *kleptography* to denote such use of *cryptography against cryptography*.

In the rest of this section, we denote the *attacker's* encryption and decryption functions by f_e and f_d respectively. We often do not restrict these functions to public-key routines only. Since public-key routines are slow, symmetric-key routines can be employed in practice. Simple XOR-ing with a fixed bit string (known to the designer) may also suffice. However, for these faster alternatives of f_e, f_d, reverse engineering reveals the symmetric key or the XOR operand to the user who can subsequently mimic the attacker to steal keys generated elsewhere by the same contaminated system.

We use the following shorthand notations. Here, n stands for a positive integer that can be naturally identified with a unique bit string having the most significant (that is, leftmost) bit equal to 1.

$$\begin{aligned} |n| &= \text{the bit length of } n. \\ \mathrm{lsb}_k(n) &= \text{the least significant } k \text{ bits of } n. \\ \mathrm{msb}_k(n) &= \text{the most significant } k \text{ bits of } n. \\ (a_1 \,||\, a_2 \,||\, \cdots \,||\, a_r) &= \text{the concatenation of the bit strings } a_1, a_2, \ldots, a_r. \end{aligned}$$

7.3.1 Attacks on RSA

RSA, (seemingly) being the most popular public-key cryptosystem, has been the target of most cryptanalytic attacks. Backdoor attacks are not an exception. The backdoor attacks on RSA work by cleverly *hiding* some secret information in the public key (n, e) of a user. As earlier, we denote the corresponding private exponent by d and the prime factors of n by p and q.

Hiding prime factor

The simplest attack is to choose a fixed p known to the designer. The other prime q is generated randomly, and correspondingly $n = pq$ and the key pairs (e, d) are computed. Reverse engineering such a scheme is pretty simple, since two different moduli $n_1 = pq_1$ and $n_2 = pq_2$ belch out $p = \gcd(n_1, n_2)$ easily.

A better approach is given in Algorithm 7.3. The function f_e may be RSA encryption under the designer's public key. In that case, the RSA modulus of the attacker should be so chosen that the condition $e < n$ is satisfied with good probability. On the other hand, if this modulus is too small, then this scheme will generate values of e much smaller than n.

In order to determine the secret exponent from a public key generated using this scheme, the attacker runs Algorithm 7.4. If f_e and f_d are RSA functions under the attacker's keys, nobody other than the attacker can apply f_d to generate p from e. This provides the designer with the exclusive capability of stealing keys.

A problem with Algorithm 7.3 is that the attacker has little control over the length of the public exponent e. If the user demands a small modulus (like $e = 3$ or $e = 257$), this

Algorithm 7.3 **A simple backdoor attack on RSA**

Input: $k \in \mathbb{N}$.
Output: An RSA modulus $n = pq$ with $|p| = |q| = k$, and exponents (e, d).
Steps:
Generate a random k-bit prime q.
while (1) {
  Generate a random k-bit prime p.
  $n := pq$.
  $e := f_e(p)$.
  if (($e < n$) and ($\gcd(e, \phi(n)) = 1$)) {
    Compute d with $ed \equiv 1 \pmod{\phi(n)}$.
    Return (n, e, d).
  }
}

Algorithm 7.4 **Retrieving the secret exponent**

Input: An RSA public key (n, e).
Output: The corresponding secret (p, q, d) or failure.
Steps:
$p := f_d(e)$.
if $(p|n)$ {
  $q := n/p$.
  $\phi := (p - 1)(q - 1)$.
  $d := e^{-1} \pmod{\phi}$.
  Return (p, q, d).
} else {
  /* The key is not generated by Algorithm 7.3 */
  Return failure.
}

scheme fails to produce one. Algorithm 7.5 overcomes this difficulty by hiding p in the high order bits of the modulus n (instead of in the exponent e). Young and Yung [307] proposed this algorithm in the name *PAP* (*pretty awful privacy*). The name contrasts with *PGP* (*pretty good privacy*), a popular and widely used RSA implementation.

Algorithm 7.5 works as follows. Following Young and Yung [307], we assume that the attacker uses RSA to realize f_e and f_d. The RSA modulus of the attacker is denoted by N. The attack requires $|N| = k$, where $|p| = |q| = k$. To start with, a random prime p of the desired bit length k is generated. This prime is to be encrypted using f_e and so one requires $p < N$. Instead of encrypting p directly, the attacker uses a permutation function π keyed by $K + i$ for some fixed K and for $i = 1, 2, \ldots, B$, where B is a small bound (typically $B = 16$). This permutation helps the attacker in two ways. First, one may now have $p > N$, so a suspicion regarding bounded values of p does not arise. Second, it is cheaper to apply the permutation instead of generating fresh candidates

for p. (In an (honest) RSA key generation routine, the prime generation part typically takes the most of the running time.)

Algorithm 7.5 **Backdoor attack on RSA: Young and Yung's PAP scheme**

Input: $k \in \mathbb{N}$.
Output: An RSA modulus $n = pq$ with $|p| = |q| = k$, and exponents (e, d).
Steps:

```
while (1) {
  /* Try to generate a suitable p */
  Generate a random k-bit prime p.
  i = 1.
  while (i ⩽ B) {
    p' := π_{K+i}(p).   /* Use a keyed permutation π_{K+i}. */
    if (p' < N) { break } else { i++ }
  }

  /* Try to generate n and q */
  if (i ⩽ B) {
    p'' := f_e(p').   /* Encrypt p' by the designer's public key */
    j := 1.
    while (j ⩽ B') {
      p''' := π'_{K+j}(p'').   /* π'_{K+j} is a keyed permutation and |p'''| = k or k − 1. */
      Generate a pseudorandom bit string a of length k.
      X := (p''' || a).
      q := X quot p.
      if (|q| = k) and (q is prime) {
        n := pq.
        e := 17.
        while (gcd(e, φ(n)) ≠ 1) { e += 2. }
        d := e^{−1} (mod φ(n)).
        Return (n, e, d).
      } else { j++ }
    }
  }
}
```

Once a suitable p and the corresponding $p' = \pi_{K+i}(p)$ are generated, the encryption function f_e is applied to generate $p'' = f_e(p')$. Now, instead of embedding p'' directly in the modulus n, another keyed permutation is applied on p'' to generate $p''' = \pi'_{K+j}(p'')$. This permutation facilitates investigating several choices for q and so is a faster alternative than restarting the entire process afresh, every time an unsuitable q is computed. A pseudorandom bit string a of length k is appended to p''' to obtain an approximation X for n. If $q := \lfloor X/p \rfloor$ happens to be a prime of bit length k, the exact $n = pq$ is computed, else another j is tried. If all values of $j \in \{1, 2, \ldots, B'\}$ (for some small bound B') fail, the entire procedure is repeated with a new k-bit prime p.

For random choices of a, the quotients $q = \lfloor X/p \rfloor$ behave like random integers and so the probability that q is prime is almost the same as random integers of bit length k.

Write $X = qp + r$ with $r = X \operatorname{rem} p$. If $r > a$, then $n = X - r$ has $p''' - 1$ embedded in its higher bits, whereas if $r \leqslant a$, then p''' itself is embedded in the higher bits of n.

Once suitable p and q are found, the PAP routine generates (like PGP) a small encryption exponent e relatively prime to $\phi(n)$ and its inverse d modulo $\phi(n)$. One can anyway opt for bigger values of e. In that case, instead of choosing e successively from the sequence $17, 19, 21, 23, \ldots$ one writes one's customized steps for generating candidate values for e. Choosing small e in Algorithm 7.5 indicates resemblance with PGP and the flexibility of doing so.

The authors of PAP compare their implementation of Algorithm 7.5 with that of the honest PGP key generation procedure. The contaminated routine has been found to run on an average only 20 per cent slower than the honest routine.

Algorithm 7.6 recovers the prime factor p of n from a public key (n, e) generated by PAP, using the RSA decryption function f_d of the attacker. Reverse engineering may make available to the user the permutation functions π and π', the fixed constants K, B, B' and the designer's public key. But this knowledge alone does not empower the user to steal PAP-generated keys.

Algorithm 7.6 **Retrieving the prime divisor**

Input: An RSA public key (n, e) with $n = pq$.
Output: The prime divisor p of n or failure.
Steps:

```
Write n = (U || V) with |V| = k.
for p''' ∈ {U, U + 1} {
   for j = 1, 2, ..., B' {
      p'' := (π'_{K+j})^{-1}(p''').
      p' := f_d(p'').
      for i = 1, 2, ..., B {
         p := (π_{K+i})^{-1}(p').
         if (p|n) { Return p. }
      }
   }
}
/* (n, e) is not generated by Algorithm 7.5 */
Return failure.
```

Hiding small private exponent

Another possible backdoor is hiding an RSA key pair (ϵ, δ) with small δ inside a key pair (e, d). Crépeau and Slakmon [70] realize this backdoor using a result from Boneh and Durfee [32], which describes a polynomial-time (in $|n|$) algorithm for computing δ from the public key (n, ϵ), provided that δ is less than $n^{0.292}$. This attack is explained in Algorithm 7.7. Here, the modulus is a genuine random RSA modulus. The mischievous key ϵ is neatly hidden by the attacker's encryption routine f_e. The resulting output key pair (e, d) looks reasonably random. However, this scheme has a drawback similar to Algorithm 7.3; that is, it cannot easily generate small values of e.

Algorithm 7.7 **Backdoor attack on RSA: small private exponent**

Input: $k \in \mathbb{N}$.
Output: An RSA modulus $n = pq$ with $|n| = k$ and a key pair (e, d).
Steps:
Generate random primes p, q of bit length $\sim k/2$, such that $n := pq$ has $|n| = k$.
do {
  Generate random $\delta \in \mathbb{Z}_n$ with $\gcd(\delta, \phi(n)) = 1$ and $|\delta| < 0.292|n|$.
  $\epsilon := \delta^{-1} \pmod{\phi(n)}$.
  $e := f_e(\epsilon)$. /* Hide ϵ */
} while ($\gcd(e, \phi(n)) \neq 1$).
$d := e^{-1} \pmod{\phi(n)}$.
Return (n, e, d).

Algorithm 7.8 retrieves d from a public key (n, e) generated by Algorithm 7.7.

Algorithm 7.8 **Retrieving the secret exponent**

Input: An RSA public key (n, e) generated by Algorithm 7.7.
Output: The corresponding private key d.
Steps:
$\epsilon := f_d(e)$. /* Recover the hidden exponent */
Use Boneh and Durfee's algorithm to recover $\delta \equiv \epsilon^{-1} \pmod{\phi(n)}$.
Use ϵ and δ to compute $\phi(n)$.
Compute $d \equiv e^{-1} \pmod{\phi(n)}$.

The correctness of Algorithm 7.8 is evident. In order to see how the knowledge of ϵ and δ reveals $\phi(n)$, note that $x := \epsilon\delta - 1$ is a multiple of $\phi(n)$; that is,

$$x = l\phi(n) = ln - l(p + q - 1) \tag{7.7}$$

for some integer l. Since $\delta < n^{0.292}$ and $\epsilon < n$, we have $x < n^{1.292}$. But $\phi(n) \approx n$ and so l cannot be much larger than $n^{0.292}$. Since $|p| \approx k/2 \approx |q|$, we have $l(p+q-1) < n$. Now, if we write

$$x = an + b = (a + 1)n - (n - b)$$

with $a = x \text{ quot } n$ and $b = x \text{ rem } n$, comparison with Equation (7.7) reveals that $l = a + 1$. This gives $\phi(n) = x/l$.

Although not needed explicitly here, the factorization of n can be easily obtained by solving the equations $pq = n$ and $p + q = n - \phi(n) + 1$. If ϵ and δ are not small, we may have $l(p + q - 1) \geqslant n$, and $\phi(n)$ cannot be calculated so easily as above. A randomized polynomial-time algorithm can still factor n from the knowledge of ϵ, δ and n. For the details, solve Exercise 7.9.

Hiding small public exponent

Crépeau and Slakmon propose another backdoor attack based on the following result due to Boneh et al. [33]. Let (ϵ, δ) be a key pair for an RSA modulus $n = pq$. Further,

let $t \in \{1, 2, \ldots, |n|/2\}$ and $2^{t-1} \leqslant \epsilon < 2^t$. There exists a polynomial-time algorithm that, given n, ϵ, and t most significant and $|n|/4$ least significant bits of δ, recovers the full private exponent δ.

Algorithm 7.9 **Backdoor attack on RSA: small public exponent**

Input: $k \in \mathbb{N}$ and $t \in \{1, 2, \ldots, \lfloor k/2 \rfloor\}$.
Output: An RSA modulus $n = pq$ with $|n| = k$ and a key pair (e, d).
Steps:
Generate random primes p, q of bit length $\sim k/2$, such that $n := pq$ has $|n| = k$.
do {
  Generate random $\epsilon \in \mathbb{Z}_n$ with $\gcd(\epsilon, \phi(n)) = 1$ and $|\epsilon| = t$.
  $\delta := \epsilon^{-1} \pmod{\phi(n)}$.
  $e := f_e\Big(\mathrm{msb}_t(\delta) \mid\mid \mathrm{lsb}_{\lceil k/4 \rceil}(\delta) \mid\mid \epsilon\Big)$.
} while ($\gcd(e, \phi(n)) \neq 1$).
$d := e^{-1} \pmod{\phi(n)}$.
Return (n, e, d).

Algorithm 7.9 uses f_e to hide in e a small ϵ, t most significant bits of δ and $|n|/4$ least significant bits of δ. A string of bit length $2t + k/4$ is encrypted by f_e. Applying the decryption routine f_d on e recovers these hidden values, from which ϵ and δ and hence $\phi(n)$ can be obtained. Algorithm 7.10 does this task. This scheme also fails, in general, to produce small public exponents e.

Algorithm 7.10 **Retrieving the secret exponent**

Input: An RSA public key (n, e) generated by Algorithm 7.9 and the matching $t \in \{1, 2, \ldots, |n|/2\}$.
Output: The corresponding private key d.
Steps:
Compute $f_d(e)$ and retrieve the following:
  (a) the hidden public exponent ϵ,
  (b) the t most significant bits of the hidden private exponent δ and
  (c) the $|n|/4$ least significant bits of δ.
Apply the Boneh-Durfee-Frankel algorithm to recover δ completely.
Use ϵ and δ to compute $\phi(n)$. /* See Exercise 7.9 */
Compute $d \equiv e^{-1} \pmod{\phi(n)}$.

7.3.2 An Attack on ElGamal Signatures

We now describe a backdoor attack on the ElGamal signature Algorithm 5.36. This attack does not work when the user's permanent key pair is generated. It manipulates the *session-key generation* in such a way that the user's permanent private key is revealed to the attacker from two successive signatures.

Let p be a prime, g a generator of $\mathbb{F}_p^*$, and $(d, g^d (\mathrm{mod}\ p))$ the permanent key pair of Alice. The attacker uses the same field and a key pair $(D, g^D (\mathrm{mod}\ p))$ with g^D supplied to the signing device. Suppose that Alice signs two messages m_1 and m_2 to generate signatures (s_1, t_1) and (s_2, t_2) respectively, where

$$\begin{aligned} s_1 &\equiv g^{d_1} \pmod p, \\ t_1 &\equiv d_1^{-1}(H(m_1) - dH(s_1)) \pmod{p-1}, \\ s_2 &\equiv g^{d_2} \pmod p, \\ t_2 &\equiv d_2^{-1}(H(m_2) - dH(s_2)) \pmod{p-1}. \end{aligned}$$

The attack proceeds by letting d_1 arbitrary, but by taking

$$d_2 \equiv (g^D)^{d_1} \pmod p.$$

Since $s_1^D \equiv (g^{d_1})^D \equiv (g^D)^{d_1} \pmod p$, we have

$$t_2 \equiv \left(s_1^D \pmod p\right)^{-1} (H(m_2) - dH(s_2)) \pmod{p-1},$$

that is,

$$d \equiv (H(s_2))^{-1} \left(H(m_2) - t_2(s_1^D \pmod p)\right) \pmod{p-1}.$$

The private key D of the attacker (or d_1) is required for computing d; so nobody other than the designer can retrieve Alice's secret by observing the contaminated signatures (s_1, t_1) and (s_2, t_2).

7.3.3 An Attack on ElGamal Encryption

For ElGamal encryption (Algorithm 5.15) and for Diffie–Hellman key exchange (Algorithm 5.27) over $\mathbb{F}_p$, a party (Alice) generates random session key pairs of the form $(d', g^{d'} (\mathrm{mod}\ p))$ and communicates the public session key $g^{d'}$ to another party. The following backdoor manipulates the session-key generation in such a way that two public session keys reveal the second private session key (but not the permanent private key). We assume that the attacker learns the public session keys by eavesdropping. The attacker's key-pair is $(D, g^D (\mathrm{mod}\ p))$. The contaminated routine contains the public key $g^D (\mathrm{mod}\ p)$, but not the private key D.

Let (d_1, r_1) and (d_2, r_2) be two session keys used by Alice, where

$$\begin{aligned} r_1 &\equiv g^{d_1} \pmod p, \\ r_2 &\equiv g^{d_2} \pmod p. \end{aligned}$$

The contaminated routine that generates the session keys uses a fixed *odd* integer u, a hash function H and a random bit $b \in \{0, 1\}$ to generate d_2 from d_1 as follows:

$$\begin{aligned} z &\equiv g^{d_1 + ub} (g^D)^{d_1} \pmod p, \\ d_2 &\equiv H(z) \pmod{p-1}. \end{aligned}$$

The attacker knows r_1 and r_2 by eavesdropping. She computes d_2 by Algorithm 7.11, the correctness of which is established from that $r_1^D \equiv (g^{d_1})^D \equiv (g^D)^{d_1} \pmod p$.

Algorithm 7.11 **Backdoor attack on ElGamal encryption**

$z_0 := r_1^{1+D} \pmod p$. /* corresponding to $b = 0$ */
if ($r_2 \equiv g^{H(z_0)} \pmod p$) { Return $H(z_0)$. }
$z_1 := z_0 g^u \pmod p$. /* corresponding to $b = 1$ */
if ($r_2 \equiv g^{H(z_1)} \pmod p$) { Return $H(z_1)$. }
Return failure. /* The attacker's routine was not used for key generation. */

Algorithm 7.11 requires the attacker's private key D (or d_1) and can be performed only by the attacker. Now, d_2 can be analogously used to generate the third session key d_3 and so on, that is, the attacker can steal all the private session keys (except the first).

The odd integer u is used for additional safety. In order to see what might happen without it (that is, with $b = 0$ always), assume that H can be inverted. This gives z and $y := zr_1^{-1} \equiv g^{Dd_1} \pmod p$. If D is even, y is always a quadratic residue modulo p. If D is odd, y is a quadratic residue or non-residue modulo p depending on whether d_1 is even or odd. The randomly added odd bias u destroys this correlation of z with quadratic residues.

7.3.4 Countermeasures

Using trustworthy implementations (hardware or software) of cryptographic routines (in particular, key generation routines) eliminates or reduces the risk of backdoor attacks. Preferences should be given to software applications with source codes (rather than to the more capable ones without source codes). Random number generators should be given specific attention. Cascading products from different independent sources also minimizes the possibility of hidden backdoors.

If the desired grain of trust is missing from the available products, the only safe alternative is to write the codes oneself. Complete trust on cryptographic devices and packages and using them as black boxes without bothering about the internals is often called *black-box cryptography*. Users should learn to question black-box cryptography. The motto is: *Be aware or bring peril.*

EXERCISE SET 7.3

7.8 Argue that reverse engineering the PAP routine (Algorithm 7.5) can enable a user to distinguish in polynomial time between key pairs generated by PAP and those generated by honest procedures.

7.9 Let $n = pq$ be an RSA modulus and (e, d) a key pair under this modulus. Write $ed - 1 = 2^s t$, where $s = \mathrm{v}_2(ed - 1)$ (so that t is odd). Since $ed - 1$ is a multiple of $\phi(n) = (p - 1)(q - 1)$ with odd p, q, we have $s \geqslant 2$.
(a) Show that for any $a \in \mathbb{Z}_n^*$ the multiplicative order $\mathrm{ord}_n(a^t)$ divides 2^s. **[H]**
(b) Let $a \in \mathbb{Z}_n^*$ be such that a^t has different orders modulo p and modulo q. Show that $\gcd(a^{2^\sigma t} - 1, n)$ is a non-trivial divisor of n for some $\sigma \in \{0, 1, \ldots, s - 1\}$.

(c) Let g be a generator of $\mathbb{Z}_p^*$. Take $a := g^k \pmod p$ for some $k \in \{0, 1, \ldots, p-2\}$ and let $\mathrm{ord}_p(a^t) = 2^\sigma$. Show that $\sigma = \mathrm{v}_2(p-1)$ if k is odd, and $\sigma < \mathrm{v}_2(p-1)$ if k is even. **[H]** An analogous result holds for the other prime q.
(d) Demonstrate that there are at least $\phi(n)/2$ elements a in $\mathbb{Z}_n^*$ with the property that a^t has different orders modulo p and q. **[H]**
(e) Suggest a randomized poly-time algorithm for factoring n from the knowledge of n, e and d.

Chapter Summary

In this chapter, we discuss some indirect ways of attacking public-key cryptosystems. These attacks do not attempt to solve the underlying intractable problems, but watch the decryption device and/or use malicious key generation routines in order to gain information about private keys.

The *timing attack* works based on the availability of the total times of several private-key operations under the same private key. It successively keeps on guessing bits of the private key by performing some variance calculations.

The *power attack* requires the availability of the power consumption patterns (also called *power traces*) of the decrypting (or signing) device during one or more private-key operations. If the measurements are done with good accuracy and resolution, a single power trace may reveal the private key to the attacker; this is called *simple power analysis*. In practice, however, such power measurements are often contaminated with noise. *Differential power analysis* requires power traces from several decryption operations under the same private key. The different traces are combined using a technique that reduces the effect of noise.

A *fault attack* can be mounted by injecting one or more faults in the device performing private-key operations. Fault attacks are discussed in connection with several encryption (RSA), signature (ElGamal, DSA and so on) and authentication (FFS) schemes.

The above three kinds of attacks are collectively called *side-channel attacks*. Several general and algorithm-specific countermeasures against side-channel attacks are discussed.

Backdoor attacks, on the other hand, are mounted by malicious key generation routines. Young and Yung propose the concept of *secretly embedded trapdoor with universal protection* (*SETUP*). In a SETUP-contaminated system, the designer of the key generation routine possesses the exclusive right to steal keys from users. Several examples of backdoor attacks on RSA and ElGamal cryptosystems are described.

Suggestions for Further Reading

Kocher introduces the concept of side-channel attacks in his seminal paper [155]. This paper describes further details about the timing attack (like a derivation of the choice of the sample size k) and some experimental results.

Timing attacks in various forms are applicable to other systems. Kocher [155] himself suggests a chosen message attack on an RSA implementation based on CRT (Algorithm 5.4). Carol, in an attempt to guess Alice's public key d, tries to guess the factor p (or q) of the modulus n using a timing attack. She starts by letting Alice sign a message y (c in Algorithm 5.4) close to an initial guess of p. The CRT-based algorithm first reduces y modulo p and modulo q before performing the modular exponentiations. If $y < p$ already, then the initial reduction modulo p returns (almost) immediately, whereas if $y \geqslant p$, the reduction involves at least one subtraction. This gives a variation in the timings based on the value of p. This fact is exploited by the attack to arrive at better and better approximations of p.

A known-message timing attack (in addition to the chosen message attack mentioned in the last paragraph) on the CRT-based RSA signature scheme is proposed by Kocher in the same paper [155]. Kocher also explains a timing attack on the signature algorithm DSA (Algorithm 5.43), based on the dependence of the modular reduction of $H(M) + ds$ modulo r on the bits of the signer's private key d.

Large scale implementations of timing attacks are reported in the technical reports [77, 259] from the Crypto group of Université catholique de Louvain. These implementations study Montgomery exponentiation.

Kocher [155] mentions the possibility of power attacks. However, a concrete description is first published in Kocher et al. [156], which explains both SPA and DPA. DES is the basic target of this paper, though possibilities for using these techniques against public-key systems are also mentioned.

Several variants of the basic DPA model described in the text have been proposed. Messerges et al. [200] describe attacks against smart-card implementations of exponentiation-based public-key systems. Also consult Aigner and Oswald's tutorial [9] for a recent survey.

DPA seems to be the most threatening of all side-channel attacks. Many papers suggesting countermeasures against DPA have appeared. Chari et al. [45] propose a masking method. Messerges [199] applies this idea to a form suitable for AES.[4] Messerges' countermeasure is broken in [63] using a multi-bit DPA. Some other useful papers on DPA include [10, 55, 201].

Boneh et al. [30, 31] from the Bellcore Lab. announce the first systematic study of fault attacks on asymmetric-key cryptosystems. They explain fault attacks on RSA (with and without CRT), the Rabin signature scheme, the Feige–Fiat–Shamir identification protocol and on the Schnorr identification protocol. These attacks are collectively known as *Bellcore attacks*.

Arjen K. Lenstra points out that the fault attack on CRT-based RSA does not require the valid signature. Joye and Quisquater propose some generalizations of the Bellcore–

[4] *AES* is an abbreviation for *advanced encryption standard* which is a US-government standard that supersedes the older standard DES. AES uses the *Rijndael cipher* [219].

Lenstra attack. A form of this attack is applicable to elliptic-curve cryptosystems. The paper [142] talks about these developments.

Bao et al. [17] propose fault attacks on DSA, ElGamal and Schnorr signatures. They also describe variants of the fault analysis of RSA based on square-and-multiply algorithms. Zheng and Matsumoto [315] indicate the possibilities of attacking the random bit generator in a smart card.

Biham and Shamir [22] investigate fault analysis of *symmetric-key* ciphers and introduce the concept of *differential fault analysis*. Anderson and Kuhn [11] also study fault analysis of symmetric-key ciphers. Aumüller et al. [15] publish their practical experiences regarding physical realizations of faults in smart cards. They also suggest countermeasures against such attacks.

James A. Muir's work [215] is a very readable and extensive survey on side-channel cryptanalysis. Also look at Boneh's survey [29].

Because of small key sizes, elliptic-curve cryptosystems are very attractive for implementation in smart cards. It is, therefore, necessary to provide effective countermeasures against side-channel attacks (most importantly, against the DPA) for elliptic-curve cryptosystems. Many recent articles discuss this issue. Coron [62] suggests the use of random projective coordinates to avoid the costly (and power-consuming) field inversion operation needed for adding and doubling of points. Möller [206] proposes a non-conventional way of carrying out the double-and-add procedure. Izu and Takagi [138] describe a Montgomery-type point addition scheme resistant against side-channel attacks. An improved version of this algorithm, that works for a more general class of elliptic curves, is presented in Izu et al. [137].

Young and Yung introduce the concept of SETUP in [307]. The PAP SETUP on RSA and the ElGamal signature SETUP are from this paper which also includes attacks on DSA and Kerberos authentication protocol. In a later paper [308], Young and Yung categorizes SETUP in three types: regular, weak and strong. Strong SETUPs are proposed for Diffie–Hellman key exchange and for RSA. The third reference [309] from the same authors extends the ideas of kleptography further and provides backdoor routines for several other cryptographic schemes.

Crépeau and Slakmon [70] adopt a more informal approach and discuss several backdoors for RSA key generation. In addition to the trapdoors with hidden small private and public exponents, described in the text, they propose a trapdoor that hides small *prime* public exponent. They also present an improved version of the PAP routine. Unlike Young and Yung, they suggest symmetric techniques for designing f_e, f_d. Symmetric techniques endanger universal protection of the attacker, but continue to make perfect sense in the context of black-box cryptography.

Quantum Computation and Cryptography

8

8.1 Introduction
8.2 Quantum Computation
8.3 Quantum Cryptography
8.4 Quantum Cryptanalysis
Chapter Summary
Sugestions for Further Reading

Our best theories are not only truer than common sense, they make far more sense than common sense does.
— David Deutsch [76]

One can be a masterful practitioner of computer science without having the foggiest notion of what a transistor is, not to mention how it works.
— N. David Mermin [197]

But suppose I could buy a truly powerful quantum computer off the shelf today — what would I do with it? I don't know, but it appears that I will have plenty of time to think about it!
— John Preskill [243]

8.1 Introduction

So far, we studied algorithms in the area of cryptology, that can be implemented on *classical* computers (Turing machines or von Neumann's stored-program computers). Now, we shift our attention to a different paradigm of computation, known as *quantum computation*. The working of a quantum computer is specified by the laws of quantum mechanics, a branch of physics developed in the 20^{th} century. However counter-intuitive, contrived or artificial these laws initially sound, they have been accepted by the physics community as robust models of certain natural phenomena. A bit, modelled as a quantum mechanical system, appears to be a more powerful unit than a classical bit to build a computing device.

This enhanced power of a computing device has many important ramifications in cryptology. On one hand, we have polynomial-time quantum algorithms to solve the integer factorization and the discrete-log problems. This implies that most of the cryptographic algorithms that we discussed earlier become (provably) insecure. On the other hand, there are proposals for a quantum key-exchange method that possesses unconditional (and provable) security.

Unfortunately, it is not clear how one can manufacture a quantum computer. Technological difficulties involved in the process appear enormous and a section of the crowd even questions the feasibility of building such a machine. However, no laws or proofs rule out the possibility of success in the (near or distant) future. Myth has it that Thomas Alva Edison, after several hundred futile attempts to manufacture an electric light bulb, asserted that he knew hundreds of ways how one cannot make an electric bulb. Edison succeeded eventually and dream turned into reality.

But we will not build quantum computers in this chapter. That is well beyond the scope of this book, or, for that matter, of computer science in general. It is thoroughly unimportant to understand the I-V curves of a transistor (or even to know what a transistor actually is), when one designs and analyses (classical) algorithms. In order to design and analyse quantum algorithms, it is equally unimportant to know how a quantum computer can be realized.

8.2 Quantum Computation

We start with a formal description of quantum computation. Quantum mechanical laws govern this paradigm. We will pay little attention to the *physical* interpretations of these laws. A mathematical formulation suffices for our purpose.

For defining a quantum mechanical *system*, we need to enrich our mathematical vocabulary. Let V be a vector space over $\mathbb{C}$ (or $\mathbb{R}$). Using *Dirac's ket notation* we denote a vector ψ in V as $|\psi\rangle$.

Definition 8.1 An *inner product* (also called a *dot product* or a *scalar product*) on V is a function $\langle \mid \rangle : V \times V \to \mathbb{C}$ satisfying the following properties:

(1) **Positivity** For any $|\psi\rangle \in V$, the inner product $\langle\psi|\psi\rangle$ is real and non-negative. Moreover, $\langle\psi|\psi\rangle = 0$ if and only if $|\psi\rangle = 0$.

(2) **Linearity** For $a_1, a_2 \in \mathbb{C}$ and $|\psi\rangle, |\varphi_1\rangle, |\varphi_2\rangle \in V$, we have $\langle\psi|a_1\varphi_1 + a_2\varphi_2\rangle = a_1\langle\psi|\varphi_1\rangle + a_2\langle\psi|\varphi_2\rangle$.

(3) **Skew symmetry** For any $|\psi\rangle, |\varphi\rangle \in V$, we have $\langle\varphi|\psi\rangle = \overline{\langle\psi|\varphi\rangle}$, where the bar denotes complex conjugate.

A vector space V with an inner product is called an *inner product space*. □

Example 8.1 For $n \in \mathbb{N}$, the space $\mathbb{C}^n$ is an inner product space with the inner product of $|\psi\rangle = (\psi_1, \ldots, \psi_n)$ and $|\varphi\rangle = (\varphi_1, \ldots, \varphi_n)$ defined as

$$\langle\psi|\varphi\rangle := \sum_{i=1}^{n} \overline{\psi}_i \varphi_i \,.$$

□

Definition 8.2 The inner product on a vector space V induces a norm (Definition 2.115) on V:

$$||\psi|| := \sqrt{\langle\psi|\psi\rangle}\,.$$

An inner product space $\mathcal{H}$ which is complete (Definition 2.119) under the norm induced by its inner product is called a *Hilbert space*. We will typically consider finite-dimensional Hilbert spaces (over $\mathbb{C}$) and for $n \in \mathbb{N}$ denote the n-dimensional Hilbert space by $\mathcal{H}_n$. □

Definition 8.3 We define an equivalence relation $\sim$ on a Hilbert space $\mathcal{H} = \mathcal{H}_n$ as $|\psi\rangle \sim |\varphi\rangle$ if and only if $\psi = a\varphi$ for some $a \in \mathbb{C} \setminus \{0\}$. An equivalence class under this relation is called a *ray* in $\mathcal{H}$. One typically considers a vector $|\psi\rangle$ with $\langle\psi|\psi\rangle = 1$ as a representative of its equivalence class. Such a representative is unique up to multiplication by complex numbers of the form $e^{\mathrm{i}\theta}$. □

Definition 8.4 An *orthonormal basis* of a Hilbert space $\mathcal{H} = \mathcal{H}_n$ is a subset B of $\mathcal{H}$ with the following properties:

(1) B is a $\mathbb{C}$-basis of $\mathcal{H}$.
(2) $\langle\psi|\psi\rangle = 1$ for every $\psi \in B$.
(3) $\langle\psi|\varphi\rangle = 0$ for every pair of distinct vectors $\psi, \varphi \in B$.

It is customary to denote the n vectors in an orthonormal basis of $\mathcal{H}_n$ by the symbols $|0\rangle, |1\rangle, \ldots, |n-1\rangle$. □

Example 8.2 $|0\rangle := (1, 0, 0 \ldots, 0),\ |1\rangle := (0, 1, 0, \ldots, 0),\ \ldots,\ |n-1\rangle := (0, 0, \ldots, 0, 1)$ form an orthonormal basis of $\mathbb{C}^n$ under the inner product of Example (8.1). □

8.2.1 System

The following axiom describes the model of a quantum mechanical system.

Axiom 8.1 **First axiom of quantum mechanics** A system is a ray in a (finite-dimensional) Hilbert space (over $\mathbb{C}$). □

Definition 8.5 The simplest non-trivial quantum mechanical system is a ray in a 2-dimensional Hilbert space $\mathcal{H}_2$. Such a system is assumed to be the basic building block of a quantum computer and is called a *quantum bit* or a *qubit*.

In order distinguish a qubit from a classical bit, we call the latter a *cbit*. □

$\mathcal{H}_2$ has an orthonormal basis $\{|0\rangle, |1\rangle\}$. In the classical interpretation, a cbit can assume only the two values $|0\rangle$ and $|1\rangle$, whereas a qubit can assume any value of the form

$$a|0\rangle + b|1\rangle \quad \text{with} \quad a, b \in \mathbb{C},\ |a|^2 + |b|^2 = 1\,.$$

Such a state of the qubit is called a *superposition* of the classical states.

Though we don't care much, at least for the moment, here are two promising candidates for realizing a qubit:

- *Spin of an electron*: The spin of a particle (like electron) in a given direction, say, along the Z-axis, is modelled as a quantum mechanical system with an orthonormal basis consisting of *spin up* and *spin down*.
- *Polarization of a photon*: Photons constitute another class of quantum systems, where the two independent states are provided by the polarization of a photon.

A conceptual example of a 2-state quantum system is the *Schrödinger cat*. The two independent states of a cat, as we classically know, are $|\text{alive}\rangle$ and $|\text{dead}\rangle$. However, if we think of the cat confined in a closed room and isolated from our observations, quantum mechanics models the state of the cat as a superposition (that is, a complex-linear combination) of these two states. But then if the quantum model were true, opening the room may reveal the cat in a non-trivial state $a|\text{alive}\rangle + b|\text{dead}\rangle$ for some complex numbers a, b with $|a|^2 + |b|^2 = 1$. It would indeed be an exciting experience. But alas, quantum mechanics precludes the possibility of such an observation. Read on to know what we would actually see, if we open the room.

8.2.2 Entanglement

A single qubit is too small to build a useful computer. We need to use several (albeit a finite number of) qubits and hence must have a way to describe the combined system in terms of the individual qubits. As the simplest and basis case, we first concentrate on combining two quantum systems into one.

Axiom 8.2 **Second axiom of quantum mechanics** Let A and B be two quantum mechanical systems with respective Hilbert spaces $\mathcal{H}_m$ and $\mathcal{H}_n$. Let $\{|i\rangle_A \mid i = 0, \ldots, m-1\}$ and $\{|j\rangle_B \mid j = 0, \ldots, n-1\}$ be orthonormal bases of these Hilbert spaces. The quantum mechanical system AB having A and B as its two parts is described by the *tensor product*

$$\mathcal{H}_{mn} = \mathcal{H}_m \otimes \mathcal{H}_n,$$

where $\mathcal{H}_{mn}$ is an mn-dimensional Hilbert space with an orthonormal basis

$$\{|i\rangle_A \otimes |j\rangle_B \mid i = 0, \ldots, m-1 \text{ and } j = 0, \ldots, n-1\}.$$

□

It is customary to abbreviate the normalized vector $|i\rangle_A \otimes |j\rangle_B$ as $|i\rangle_A|j\rangle_B$ or even as $|ij\rangle_{AB}$. A general state of AB is of the form

$$\sum_{i,j} a_{ij}|ij\rangle \quad \text{with} \quad \sum_{i,j} |a_{ij}|^2 = 1\,.$$

We can generalize this construction to describe a system having $k \in \mathbb{N}$ components $A_1, \ldots, A_k$. If $\mathcal{H}_{n_i}$ is the Hilbert space of A_i with an orthonormal basis $\{|j\rangle_i \mid 0 \leqslant j < n_i\}$, the composite system $A_1 \cdots A_k$ has the $n_1 \cdots n_k$-dimensional Hilbert space with an orthonormal basis comprising the vectors

$$|j_1\rangle_1 \otimes |j_2\rangle_2 \otimes \cdots \otimes |j_k\rangle_k = |j_1\rangle_1|j_2\rangle_2 \cdots |j_k\rangle_k = |j_1 j_2 \ldots j_k\rangle$$

with $0 \leqslant j_i < n_i$ for all $i = 1, \ldots, k$.

Definition 8.6 An n-bit *quantum register* is a system having exactly n qubits. □

Let $A_1, \ldots, A_n$ denote the individual bits in an n-bit quantum register A. Each A_i has the Hilbert space $\mathcal{H}_2$ with orthonormal basis $\{|0\rangle, |1\rangle\}$. So A has the 2^n-dimensional Hilbert space $\mathcal{H}_{2^n}$ with an orthonormal basis consisting of the vectors

$$|j_1\rangle \otimes |j_2\rangle \otimes \cdots \otimes |j_n\rangle = |j_1\rangle|j_2\rangle \cdots |j_n\rangle = |j_1 j_2 \ldots j_n\rangle$$

with each $j_i \in \{0, 1\}$. Viewed as an integer in binary notation, $j_1 j_2 \ldots j_n$ is an integral value between 0 and $2^n - 1$. This gives us a canonical numbering $|0\rangle, |1\rangle, \ldots, |2^n - 1\rangle$ of the basis vectors for the register A. These 2^n values are precisely the states that a classical n-bit register can have. The quantum register can, however, be in any state $|\psi\rangle$ which is a *superposition* of the classical states:

$$|\psi\rangle = \sum_{i=0}^{2^n-1} a_i|i\rangle, \quad a_i \in \mathbb{C}, \quad \sum_{i=0}^{2^n-1} |a_i|^2 = 1\,.$$

Let us once again look at the general composite system $A = A_1 \cdots A_k$. In the classical sense, each state of A is composed of the individual states of the subsystems A_i. For example, each of the 2^n classical states of an n-bit register corresponds to a choice between $|0\rangle$ and $|1\rangle$ for each individual bit. That is, each individual component retains its own state in a classical composite system. This is, however, not the case with a quantum composite system. Just think of a 2-bit quantum register $C := AB$. A state

$$|\psi\rangle_C = c_0|0\rangle_C + c_1|1\rangle_C + c_2|2\rangle_C + c_3|3\rangle_C$$

of C equals a tensor product

$$\begin{aligned} |\psi_1\rangle_A \otimes |\psi_2\rangle_B &= (a_0|0\rangle_A + a_1|1\rangle_A) \otimes (b_0|0\rangle_B + b_1|1\rangle_B) \\ &= a_0b_0|0\rangle_C + a_0b_1|1\rangle_C + a_1b_0|2\rangle_C + a_1b_1|3\rangle_C, \end{aligned}$$

if and only if $c_0c_3 = c_1c_2$.

Definition 8.7 The state $|\psi\rangle$ of a quantum register $A = A_1 \cdots A_n$ is called *entangled*, if $|\psi\rangle$ cannot be written as a tensor product of the states of any two parts of A. In other others, $|\psi\rangle$ is entangled if and only if no set of fewer than n qubits of A possesses its individual state. □

Entanglement essentially implies correlation or interaction between the components. In a composite quantum system, we cannot treat the components individually. A quantum system, as we have defined (axiomatically) earlier, is a completely isolated system. In reality, interactions with the surroundings make a (non-isolated) system change its state and get entangled. This is one of the biggest problems in the realization of a quantum computer. Quantum error correction is an important topic in quantum computation. For our purpose, we stick to the abstract model of an isolated system (quantum register) immune from external disturbances.

8.2.3 Evolution

Quantum registers give us a way to store quantum information. A computation involves manipulating the information stored in the registers. In quantum mechanics, all such operations must be *reversible*, that is, it must be possible to invert every operation. The only invertible operations on the classical states $|0\rangle, |1\rangle, \ldots, |2^n - 1\rangle$ of an n-bit quantum register A are precisely all the permutations of the classical states. Now that A can be in many more (quantum) states, there are other allowed operations on A. Any such operation must be reversible and of a particular type. This is the third axiom of quantum mechanics, which is detailed shortly.

A classical n-bit register supports many non-invertible operations. For example, erasing the content of the register (that is, resetting all the bits to zero) is a non-invertible process, since the pre-erasure state of the register cannot be uniquely determined after the erase operation is carried out. Classical computation is based on (classical) gates (like NOT, AND, OR, XOR, NOR, NAND), most of which are non-invertible. XOR, as an example, requires two input bits and outputs a single bit. It is impossible to determine the inputs uniquely from the output only. All such non-reversible operations are disallowed in the quantum world. An invertible version of the XOR operation takes two bits x and y as input and outputs the two bits x and $x \oplus y$ (where $\oplus$ denotes XOR of bits). Given the output $(x, x \oplus y)$, the input can be uniquely determined as $(x, y) = (x, x \oplus (x \oplus y))$, that is, by applying the reversible XOR operation once more.

Like XOR, all bit operations that build up a classical computer can be realized using reversible operations only. This gives us the (informal) assurance that quantum computers are at least as powerful as classical computers.

Back to the business—the third axiom of quantum mechanics.

Definition 8.8 Let U be a square matrix (that is, an $m \times m$ matrix for some $m \in \mathbb{N}$) with complex entries. The conjugate transpose of U is denoted by the symbol $U^\dagger$, that is, if $U = (u_{ij})$, then $U^\dagger = (\overline{u_{ji}})$. U is called *unitary*, if $UU^\dagger = U^\dagger U = I$, where I is the $m \times m$ identity matrix. Every unitary matrix U is invertible with $U^{-1} = U^\dagger$, and preserves the inner product of $\mathbb{C}^m$, that is, $\langle U\psi|U\varphi\rangle = \langle\psi|\varphi\rangle$ for $|\psi\rangle, |\varphi\rangle \in \mathbb{C}^m$. □

Let A be a quantum system (like a quantum register) with Hilbert space $\mathcal{H}_m$. An $m \times m$ unitary matrix U defines a *unitary linear transformation* on $\mathcal{H}_m$ taking a normalized vector $|\psi\rangle$ to a normalized vector $U|\psi\rangle$. Moreover, the transformation maps an orthonormal basis of $\mathcal{H}_m$ to another orthonormal basis of $\mathcal{H}_m$ (Exercise 8.4).

Axiom 8.3 **Third axiom of quantum mechanics** A quantum system evolves unitarily, that is, any operation on a quantum mechanical system is a unitary transformation. □

Example 8.3 The *Hadamard transform* H on one qubit is defined as:

$$\begin{aligned} H|0\rangle &= \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle), \\ H|1\rangle &= \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle). \end{aligned}$$

(Recall that a linear transformation is completely specified by its images of the elements of a basis.) If one takes $|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ and $|1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$, the Hadamard transform corresponds to the unitary matrix

$$\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}.$$

By linearity, H transforms a general state $|\psi\rangle = a|0\rangle + b|1\rangle$ to the state

$$H|\psi\rangle = aH|0\rangle + bH|1\rangle = \left(\frac{a+b}{\sqrt{2}}\right)|0\rangle + \left(\frac{a-b}{\sqrt{2}}\right)|1\rangle.$$

Some other unitary operators are described in Exercises 8.5 and 8.6. □

An important consequence of quantum mechanical dynamics is that *cloning of a state of a system is not permissible*. In other words, there does not exist an operator that copies an arbitrary state (content) of one quantum register to another.

Theorem 8.1 **No-cloning theorem** For two n-bit registers A and B, there do *not* exist a unitary transform U of the composite system AB and a state $|\varphi\rangle$ of B, such that $U|\psi\rangle|\varphi\rangle = |\psi\rangle|\psi\rangle$ for *every* state of $|\psi\rangle$ of A.

Proof Assume that such a state $|\varphi\rangle$ of B and a unitary transform U of AB exist. Take two states $|\psi_1\rangle$ and $|\psi_2\rangle$ of A. Then, $U|\psi_1\rangle|\varphi\rangle = |\psi_1\rangle|\psi_1\rangle$ and $U|\psi_2\rangle|\varphi\rangle = |\psi_2\rangle|\psi_2\rangle$. By linearity, we have $U|a\psi_1 + b\psi_2\rangle|\varphi\rangle = a|\psi_1\rangle|\psi_1\rangle + b|\psi_2\rangle|\psi_2\rangle$. Now, since U clones $|a\psi_1 + b\psi_2\rangle$ also, $U|a\psi_1 + b\psi_2\rangle|\varphi\rangle = |a\psi_1 + b\psi_2\rangle|a\psi_1 + b\psi_2\rangle = a^2|\psi_1\rangle|\psi_1\rangle + b^2|\psi_2\rangle|\psi_2\rangle + ab|\psi_1\rangle|\psi_2\rangle + ab|\psi_2\rangle|\psi_1\rangle$. The two expressions for $U|a\psi_1 + b\psi_2\rangle|\varphi\rangle$ are different, unless $a = 0, b = 1$ or $a = 1, b = 0$. ■

8.2.4 Measurement

We have seen how to represent a quantum mechanical system and do operations on the system. Now comes the final part of the game, namely observing or measuring or reading the state of a quantum system. In classical computation, reading the value stored in a classical register is a trivial exercise—just read it! In quantum mechanics, this is not the case.

Axiom 8.4 **Fourth axiom of quantum mechanics—the Born rule** Let A be a quantum mechanical system with an orthonormal basis $\{|0\rangle, |1\rangle, \ldots, |m-1\rangle\}$. Assume that A is in a state $|\psi\rangle = \sum_{i=0}^{m-1} a_i|i\rangle$. A *measurement* of A at this state is a mechanism (or device) that outputs one of the integers $i \in \{0, 1, \ldots, m-1\}$, and i is output with

probability $|a_i|^2$. If i is output by the measurement, the system *collapses* from the state $|\psi\rangle$ to the state $|i\rangle$ after the measurement. □

This means that whatever the state $|\psi\rangle$ of A was before the measurement, the process of measurement can reveal only one of m possible integer values. Moreover, the measurement causes a total loss of information about the pre-measurement amplitudes a_i. Thus, it is impossible to measure A repeatedly at the state $|\psi\rangle$ to see a statistical pattern in the occurrences of different values of i so as to guess the probabilities $|a_i|^2$.

If we open *the* room, we can see the Schrödinger cat in only one of the two possible states: $|\text{alive}\rangle$ or $|\text{dead}\rangle$. Well, then, what else can we expect? Quantum mechanics only models the cat in the isolated room as one evolving following the unitary dynamics.

At first glance, this is rather frustrating. We claim that the system went through a series of classically meaningless states, but the classical states are all we can see. What is the guarantee that the system really evolved in the quantum mechanical way? Well, there is no guarantee actually. The solace is that the axioms of quantum mechanics can explain certain natural phenomena. Also it is perfectly consistent with the classical behaviour in that if the system A evolves classically and is measured at the state $|i\rangle$ (so that $a_i = 1$ and $a_j = 0$ for $j \neq i$), measuring A reveals i with probability one and causes the system to collapse to the state $|i\rangle$, that is, to remain in the state $|i\rangle$ itself.

There is a positive side of the quantum mechanical axioms. A quantum mechanical system is *inherently parallel*. An n-bit classical register at any point of time can hold only one of the classical values $|0\rangle, \ldots, |2^n - 1\rangle$. An n-bit quantum register, on the other hand, can *simultaneously* hold *all* these classical values, with respective probabilities. This inherent parallelism seems to impart a good deal of power to a computing device. Of course, as long as we cannot harness some physical objects to build a real quantum mechanical computing device, quantum computation continues to remain *science fiction*. But on an algorithmic level, the inherent parallelism of a (hypothetical) quantum computer can be exploited to do miracles, for example, to design a polynomial-time integer factorization algorithm. This is where we win—at least conceptually. Our failure to see a cat in the state $\frac{1}{\sqrt{2}}(|\text{alive}\rangle - |\text{dead}\rangle)$ should not bother us at all!

Measurement of a quantum register gives us a way to *initialize* a quantum register A to a state $|\psi\rangle$. Suppose that we get the value i upon measuring A. We then apply *any* unitary transform on A that changes A from the post-measurement state $|i\rangle$ to the desired state $|\psi\rangle$.

The measurement described in Axiom 8.4 is called *measurement in the classical basis*. The system A has, in general, many orthonormal bases other than the classical one $\{|0\rangle, \ldots, |m-1\rangle\}$. If B is any such basis, we can conceive of measuring A in the basis B. All we need to perform is to *rewrite* the state of A in terms of the new basis B. This can be achieved by applying to A a unitary transformation (the change-of-basis transformation) before the measurement in the classical basis is carried out.

A generalization of the Born rule is also worth mentioning here. Suppose that we have an $m+n$-bit quantum register A and we want to measure not all but some of the bits of A. To be more specific, let us say that we want to measure the leftmost m bits of A, though the generalized Born rule works for any arbitrary choice of m bit positions in the register A. Denoting by $|i\rangle_m$, $i = 0, \ldots, 2^m - 1$, the canonical basis vectors for

the left m bits and by $|j\rangle_n$, $j = 0, \ldots, 2^n - 1$, those for the right n bits, a general state of A can be written as

$$|\psi\rangle_{m+n} = \sum_{i,j} a_{i,j}|i,j\rangle_{m+n}$$

with $\sum_{i,j} |a_{i,j}|^2 = 1$ and with $|i,j\rangle_{m+n}$ identified as $|i\rangle_m|j\rangle_n = |i\rangle_m \otimes |j\rangle_n$. A measurement of the left m bits of A yields an integer i, $0 \leqslant i \leqslant 2^m - 1$, with probability $p_i := \sum_{j=0}^{2^n-1} |a_{i,j}|^2$. Also this measurement causes A to collapse to the state $|i\rangle_m \otimes (\frac{1}{\sqrt{p_i}} \sum_j a_{i,j}|j\rangle_n)$.

Now, if we immediately apply the generalized Born rule once again on the right n bits of A, we get an integer j, $0 \leqslant j \leqslant 2^n - 1$, with probability $|a_{i,j}|^2/p_i$ and the system collapses to the state $|i\rangle_n|j\rangle_n$. The probability of getting $|i\rangle_n|j\rangle_n$ by this two step process is then $p_i|a_{i,j}|^2/p_i = |a_{i,j}|^2$. This is consistent with a single application of the original Born rule.

8.2.5 The Deutsch Algorithm

We start with a general framework of doing computations using quantum registers. Suppose we want to compute a function f which requires an m-bit integer as input and which outputs an n-bit integer. A general function f need not be invertible, but we cannot afford non-invertible operations on quantum registers. This is why we work on an $m + n$-bit quantum register A in which the left m bits represent the input and the right n bits the output. Computing $f(x)$ for a given x is tantamount to designing a unitary transformation U_f that acts on A and converts its state from $|x\rangle_m|y\rangle_n$ to $|x\rangle_m|f(x) \oplus y\rangle_n$, where $\oplus$ is the bitwise XOR operation, and where the subscripts (m and n) indicate the number of bits in the input or output part of A. It is easy to verify that U_f is unitary. Moreover, the inverse of U_f is U_f itself. For $y = 0$, we, in particular, have $U_f(|x\rangle_m|0\rangle_n) = |x\rangle_m|f(x)\rangle_n$.

It may still be unclear to the reader what one really gains by using this quantum model. The answer lies in the parallelism inherent in a quantum register. In order to see how this parallelism can be exploited, we describe David Deutsch's algorithm which, being the first known quantum algorithm, has enough historical importance to be included here in spite of its apparent irrelevance in the context of cryptology.

Assume that $f : \{0,1\} \to \{0,1\}$ is a function that operates on one bit and outputs one bit. There are four such functions: Two of these are constant functions ($f(0) = f(1)$) and the remaining two non-constant ($f(0) \neq f(1)$). We are given a black box D_f representing f. We don't know which one of the four functions D_f actually implements, but we can supply a bit to D_f as input and read its output on this bit. Our task is to determine whether D_f represents a constant function or not. Classically, we make two invocations of D_f on the inputs 0 and 1 and make a comparison of the output values $f(0)$ and $f(1)$. It is impossible to solve the problem classically using only one invocation of the black box. The Deutsch algorithm makes this task possible using quantum computational techniques.

Following the general quantum computational model we assume that D_f is a unitary transformation on a 2-bit register A (with $m = n = 1$) that computes $D_f|x\rangle|y\rangle = |x\rangle|f(x) \oplus y\rangle$ with the left (resp. the right) bit corresponding to the input (resp. the

output) of f. Instead of supplying a classical input to D_f we initialize the register A to the state

$$\left(\frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle\right)\left(\frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle\right) = \frac{1}{2}(|0\rangle|0\rangle - |0\rangle|1\rangle - |1\rangle|0\rangle + |1\rangle|1\rangle).$$

Linearity shows that on this input, D_f ends its execution leaving A in the state

$$\begin{cases} \frac{1}{2}\left(|0\rangle - |1\rangle\right)\left(|f(0)\rangle - |\bar{f}(0)\rangle\right) & \text{if } f(0) = f(1), \\ \frac{1}{2}\left(|0\rangle + |1\rangle\right)\left(|f(0)\rangle - |\bar{f}(0)\rangle\right) & \text{if } f(0) \neq f(1). \end{cases}$$

Here, $\bar{f}(x) = 1 \oplus f(x) = 1 - f(x)$. We won't measure A right now, but apply the Hadamard transform on the left bit. This transforms A to the state

$$\begin{cases} |1\rangle\frac{1}{\sqrt{2}}\left(|f(0)\rangle - |\bar{f}(0)\rangle\right) & \text{if } f(0) = f(1), \\ |0\rangle\frac{1}{\sqrt{2}}\left(|f(0)\rangle - |\bar{f}(0)\rangle\right) & \text{if } f(0) \neq f(1). \end{cases}$$

Now, if we measure the input bit, we deterministically get the integer 1 or 0 according as whether f is constant or not respectively. That's it!

Deutsch's algorithm solved a rather artificial problem, but it opened up the possibilities of exploring a new paradigm of computation. Till date, (good) quantum algorithms are known for many interesting computational problems. In the rest of this chapter, we concentrate on some of the quantum algorithms that have an impact in cryptology.

EXERCISE SET 8.2

8.1 Let S be a finite set and let $l^2(S)$ denote the set of all functions $\psi : S \to \mathbb{C}$.
(a) Show that $l^2(S)$ is a Hilbert space under the inner product

$$\langle\psi|\varphi\rangle := \sum_{x\in S} \overline{\psi(x)}\varphi(x)\,.$$

(b) Let $B := \{\delta_x \in l^2(S) \mid x \in S\}$, where $\delta_x(y)$ is 1 if $y = x$, and is 0 otherwise. Show that B is an orthonormal basis of $l^2(S)$.

8.2 Show that the vectors $\frac{1}{\sqrt{2}}\begin{pmatrix}1\\1\end{pmatrix}$ and $\frac{1}{\sqrt{2}}\begin{pmatrix}1\\-1\end{pmatrix}$ form an orthonormal $\mathbb{C}$-basis of $\mathbb{C}^2$.

8.3 Show that $\frac{1}{\sqrt{2}}(|01\rangle - |10\rangle)$ is an entangled state of a 2-bit quantum register.

8.4 Prove the following assertions.
(a) The matrix $\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1 & 0\\ -\mathrm{i} & \mathrm{i} & 0\\ 0 & 0 & \mathrm{i}\sqrt{2}\end{pmatrix}$ is unitary.
(b) A unitary matrix preserves inner product, that is, if U is an $m \times m$ unitary matrix and $|\psi\rangle, |\varphi\rangle \in \mathbb{C}^m$, then $\langle U\psi|U\varphi\rangle = \langle\psi|\varphi\rangle$.

(c) The determinant of a unitary matrix has absolute value 1.
(d) Every eigen value of a unitary matrix has absolute value 1.
(e) An $m \times m$ matrix A is unitary if and only if the columns of A constitute an orthonormal basis of $\mathbb{C}^m$ (over $\mathbb{C}$).

8.5 **(a)** Show that the following operators are unitary on a qubit. Also construct the corresponding transformation matrices.

Identity operator	$I\|0\rangle = \|0\rangle$, $I\|1\rangle = \|1\rangle$.
Exchange operator	$X\|0\rangle = \|1\rangle$, $X\|1\rangle = \|0\rangle$.
Z operator	$Z\|0\rangle = \|0\rangle$, $Z\|1\rangle = -\|1\rangle$.
Hadamard operator	$H\|0\rangle = \frac{1}{\sqrt{2}}(\|0\rangle + \|1\rangle)$, $H\|1\rangle = \frac{1}{\sqrt{2}}(\|0\rangle - \|1\rangle)$.

(b) Deduce the following identities:

$$\begin{aligned} X^2 &= I \\ Z^2 &= I \\ H^2 &= I \\ XZ &= -ZX \\ H &= \frac{1}{\sqrt{2}}(X+Z) \\ HX &= ZH \\ HXH &= Z \\ HZH &= X \end{aligned}$$

(c) Let $\sqrt{X} = \frac{1}{1+\mathrm{i}} \begin{pmatrix} 1 & \mathrm{i} \\ \mathrm{i} & 1 \end{pmatrix}$. Show that $\sqrt{X}$ defines a unitary operator on a qubit and that $(\sqrt{X})^2 = X$, where the last X is the matrix of the exchange operator.

8.6 Let A be an n-bit quantum register. Let us plan to number the bits of A as $1, \ldots, n$ from left to right. One can apply the operators like X, Z, H of Exercise 8.5 on each individual bit of A. A qubit operation B applied on bit i of A will be denoted by B_i.
(a) Let S_{ij} be the operator that swaps bit i with bit j. Show that

$$S_{ij} = \frac{1}{2}(I + Z_iZ_j) + X_iX_j\frac{1}{2}(I - Z_iZ_j).$$

(b) Let C be the reversible XOR operation (also called the *controlled-NOT* operation) on a two-bit register $A = (A_1A_2)$, that is, $C|xy\rangle = |x\rangle|x \oplus y\rangle$. Show that C can be realized as

$$C = \frac{1}{2}(I + Z_1) + X_2\frac{1}{2}(I - Z_1) = \frac{1}{2}(I + Z_1 + X_2 - X_2Z_1).$$

8.7 Suppose that whenever you switch on your quantum computer, every bit in its registers is initialized to the state $|0\rangle$. Describe how you can use the operators I, X, Z and H defined in Exercise 8.5, in order to change the state of a qubit from $|0\rangle$ to the following:
(a) $|1\rangle$ **(b)** $-|1\rangle$ **(c)** $\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ **(d)** $\frac{1}{\sqrt{2}}(|1\rangle - |0\rangle)$ **(e)** $-\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$

8.8 Let A be an n-bit quantum register at the state $|0\rangle_n$. Show that the application of the Hadamard transform individually to each bit of A transforms A to the state $|\psi\rangle = \frac{1}{2^{n/2}} \sum_{i=0}^{2^n-1} |i\rangle_n$. This is precisely the state of A in which all of the 2^n possible outcomes in a measurement of A are equally likely. What happens if we apply H a second time individually to each bit of A, that is, what is $H_1 H_2 \cdots H_n |\psi\rangle$, where H_i denotes the Hadamard transform on the i-th bit of A?

8.9 We know that any arithmetic or Boolean operation can be implemented using AND and NOT gates. This exercise suggests a reversible way to implement these operations.
The *Toffoli gate* is a function $T : \{0,1\}^3 \to \{0,1\}^3$ that maps $(x, y, z) \mapsto (x, y, z \oplus xy)$, where $\oplus$ means XOR, and xy means AND of x and y. Thus, T flips the third bit, if and only if the first two bits are both 1.
(a) Show that T is a unitary transformation on a 3-bit quantum register. What is the inverse of T?
(b) Use T to realize the Boolean AND and NOT operations.

8.3 Quantum Cryptography

We now describe the quantum key-exchange algorithm due to Bennett and Brassard. The original paper also talks about a practical implementation of the algorithm—an implementation using polarization of photons. For this moment, we do not highlight such specific implementation issues, but describe the algorithm in terms of the conceptual computational units called qubits.

The usual actors Alice and Bob want to agree upon a shared secret using communication over an insecure channel. A third party who gave her name as Carol plans to eavesdrop during the transmission. Alice and Bob repeat the following steps. Here, H stands for the Hadamard transform.

Algorithm 8.1 **Quantum key-exchange algorithm**

Alice generates a random classical bit $i \in \{0,1\}$.
Alice makes a random choice $x \in \{0,1\}$.
Alice computes the quantum bit $A := H^x |i\rangle$.
Alice sends A to Bob.
Bob makes a random choice $y \in \{0,1\}$.
Bob computes $B := H^y A$.
Bob measures B to get the classical bit $j \in \{0,1\}$.
Bob sends y to Alice.
Alice sends x to Bob.
if ($x = y$) { Bob and Alice retains $i = j$ }

The algorithm works as follows. Alice generates a random bit i and a random decision x whether she is going to use the Hadamard transform H. If $x = 0$, she sends the quantum bit $|0\rangle$ or $|1\rangle$ to Bob. If $x = 1$, she sends either $H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ or $H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ to Bob. At this point Bob does not know whether Alice applied

H before the transmission. So Bob makes a random guess $y \in \{0, 1\}$ and accordingly skips/applies the Hadamard transform on the qubit received. If $x = y = 0$, then Bob has the qubit $B = H^0 H^0 |i\rangle = |i\rangle$ and a measurement of this qubit reveals i with probability 1. On the other hand, if $x = y = 1$, then $B = H^2 |i\rangle = |i\rangle$, since H^2 is the identity transform (Exercise 8.5). In this case also, Bob retrieves Alice's classical bit i with certainty by measuring B.

If $x \neq y$, then B is generated from Alice's initial choice $|i\rangle$ using a single application of H, that is, $B = \frac{1}{\sqrt{2}}(|0\rangle \pm |1\rangle)$ in this case. A measurement of this bit outputs 0 or 1, each with probability $\frac{1}{2}$, that is, Bob gathers no idea about the initial choice of Alice. So after it is established that $x \neq y$, they both discard the bit.

If we assume that x and y are uniformly chosen, Bob and Alice succeed in having $x = y$ about half of the time. They eventually set up an n-bit secret after about $2n$ invocations of the above protocol. Table 8.1 illustrates a sample session between Alice and Bob. After 20 iterations of the above procedure, they agree upon the shared secret 0001110111.

Table 8.1 A sample session of the quantum key-exchange algorithm

Iteration	i	x	A	y	B	j	Common bit
1	0	1	$\frac{1}{\sqrt{2}}(\|0\rangle + \|1\rangle)$	0	$\frac{1}{\sqrt{2}}(\|0\rangle + \|1\rangle)$	1	
2	0	0	$\|0\rangle$	1	$\frac{1}{\sqrt{2}}(\|0\rangle + \|1\rangle)$	1	
3	0	1	$\frac{1}{\sqrt{2}}(\|0\rangle + \|1\rangle)$	1	$\|0\rangle$	0	0
4	0	1	$\frac{1}{\sqrt{2}}(\|0\rangle + \|1\rangle)$	0	$\frac{1}{\sqrt{2}}(\|0\rangle + \|1\rangle)$	0	
5	1	1	$\frac{1}{\sqrt{2}}(\|0\rangle - \|1\rangle)$	0	$\frac{1}{\sqrt{2}}(\|0\rangle - \|1\rangle)$	1	
6	0	0	$\|0\rangle$	0	$\|0\rangle$	0	0
7	0	0	$\|0\rangle$	0	$\|0\rangle$	0	0
8	1	0	$\|1\rangle$	0	$\|1\rangle$	1	1
9	0	0	$\|0\rangle$	1	$\frac{1}{\sqrt{2}}(\|0\rangle + \|1\rangle)$	0	
10	1	1	$\frac{1}{\sqrt{2}}(\|0\rangle - \|1\rangle)$	0	$\frac{1}{\sqrt{2}}(\|0\rangle - \|1\rangle)$	0	
11	0	1	$\frac{1}{\sqrt{2}}(\|0\rangle + \|1\rangle)$	0	$\frac{1}{\sqrt{2}}(\|0\rangle + \|1\rangle)$	1	
12	0	0	$\|0\rangle$	1	$\frac{1}{\sqrt{2}}(\|0\rangle + \|1\rangle)$	0	
13	1	0	$\|1\rangle$	1	$\frac{1}{\sqrt{2}}(\|0\rangle - \|1\rangle)$	1	
14	1	1	$\frac{1}{\sqrt{2}}(\|0\rangle - \|1\rangle)$	1	$\|1\rangle$	1	1
15	1	1	$\frac{1}{\sqrt{2}}(\|0\rangle - \|1\rangle)$	1	$\|1\rangle$	1	1
16	0	1	$\frac{1}{\sqrt{2}}(\|0\rangle + \|1\rangle)$	1	$\|0\rangle$	0	0
17	1	1	$\frac{1}{\sqrt{2}}(\|0\rangle - \|1\rangle)$	1	$\|1\rangle$	1	1
18	1	0	$\|1\rangle$	0	$\|1\rangle$	1	1
19	0	1	$\frac{1}{\sqrt{2}}(\|0\rangle + \|1\rangle)$	0	$\frac{1}{\sqrt{2}}(\|0\rangle + \|1\rangle)$	0	
20	1	0	$\|1\rangle$	0	$\|1\rangle$	1	1

What remains to explain is how this protocol guards against eavesdropping by Carol. Let us model Carol as a passive adversary who intercepts the qubit A transmitted by Alice, investigates the bit to learn about Alice's secret i and subsequently transmits the qubit to Bob. In order to guess i, Carol mimics the role of Bob. At this point

Carol does not know x, so she makes a guess z about x, accordingly skips/applies the Hadamard transform on the intercepted qubit in order to get a qubit C, measures C to get a bit value k and sends the measured qubit D to Bob. (Recall from Theorem 8.1 that it is impossible for Carol to make a copy of A, work on this copy and transmit the original qubit A to Bob.) Bob receives D, assumes that it is the qubit A transmitted by Alice and carries out his part of the work to generate the bit j. Bob and Alice *later* reveal x and y. If $x \neq y$, they anyway reject the bits obtained from this iteration. Carol should also reject her bit k in this case. So let us concentrate only on the case that $x = y$. The introduction of Carol in the protocol changes A to D and hence Alice and Bob may eventually agree upon distinct bits. A sample session of the protocol in presence of Carol is illustrated in Table 8.2. The three parties generate the secret as:

Alice	0110 0111 1000 1011
Bob	0101 1101 1100 1011
Carol	0100 0101 0100 1011

Table 8.2 Eavesdropping during a key-exchange session

Iteration	i	x	A	z	$C = H^z A$	k	D	y	$B = H^y D$	j
1	0	1	$\frac{1}{\sqrt{2}}(\|0\rangle + \|1\rangle)$	1	$\|0\rangle$	0	$\|0\rangle$	1	$\frac{1}{\sqrt{2}}(\|0\rangle + \|1\rangle)$	0
2	1	0	$\|1\rangle$	0	$\|1\rangle$	1	$\|1\rangle$	0	$\|1\rangle$	1
3	1	0	$\|1\rangle$	1	$\frac{1}{\sqrt{2}}(\|0\rangle - \|1\rangle)$	0	$\|0\rangle$	0	$\|0\rangle$	0
4	0	1	$\frac{1}{\sqrt{2}}(\|0\rangle + \|1\rangle)$	0	$\frac{1}{\sqrt{2}}(\|0\rangle + \|1\rangle)$	0	$\|0\rangle$	1	$\frac{1}{\sqrt{2}}(\|0\rangle + \|1\rangle)$	1
5	0	1	$\frac{1}{\sqrt{2}}(\|0\rangle + \|1\rangle)$	1	$\|0\rangle$	0	$\|0\rangle$	1	$\frac{1}{\sqrt{2}}(\|0\rangle + \|1\rangle)$	1
6	1	1	$\frac{1}{\sqrt{2}}(\|0\rangle - \|1\rangle)$	1	$\|1\rangle$	1	$\|1\rangle$	1	$\frac{1}{\sqrt{2}}(\|0\rangle - \|1\rangle)$	1
7	1	1	$\frac{1}{\sqrt{2}}(\|0\rangle - \|1\rangle)$	0	$\frac{1}{\sqrt{2}}(\|0\rangle - \|1\rangle)$	0	$\|0\rangle$	1	$\frac{1}{\sqrt{2}}(\|0\rangle + \|1\rangle)$	0
8	1	0	$\|1\rangle$	0	$\|1\rangle$	1	$\|1\rangle$	0	$\|1\rangle$	1
9	1	1	$\frac{1}{\sqrt{2}}(\|0\rangle - \|1\rangle)$	0	$\frac{1}{\sqrt{2}}(\|0\rangle - \|1\rangle)$	0	$\|0\rangle$	1	$\frac{1}{\sqrt{2}}(\|0\rangle + \|1\rangle)$	1
10	0	1	$\frac{1}{\sqrt{2}}(\|0\rangle + \|1\rangle)$	0	$\frac{1}{\sqrt{2}}(\|0\rangle + \|1\rangle)$	1	$\|1\rangle$	1	$\frac{1}{\sqrt{2}}(\|0\rangle - \|1\rangle)$	1
11	0	0	$\|0\rangle$	1	$\frac{1}{\sqrt{2}}(\|0\rangle + \|1\rangle)$	0	$\|0\rangle$	0	$\|0\rangle$	0
12	0	0	$\|0\rangle$	0	$\|0\rangle$	0	$\|0\rangle$	0	$\|0\rangle$	0
13	1	1	$\frac{1}{\sqrt{2}}(\|0\rangle - \|1\rangle)$	1	$\|1\rangle$	1	$\|1\rangle$	1	$\frac{1}{\sqrt{2}}(\|0\rangle - \|1\rangle)$	1
14	0	0	$\|0\rangle$	0	$\|0\rangle$	0	$\|0\rangle$	0	$\|0\rangle$	0
15	1	0	$\|1\rangle$	0	$\|1\rangle$	1	$\|1\rangle$	0	$\|1\rangle$	1
16	1	0	$\|1\rangle$	1	$\frac{1}{\sqrt{2}}(\|0\rangle - \|1\rangle)$	1	$\|1\rangle$	0	$\|1\rangle$	1

In this example, Alice and Bob's shared secrets differ in five bit positions. Carol's intervention causes a shared bit to differ with a probability of $\frac{3}{8}$ (Exercise 8.11). Thus, the more Carol eavesdrops, the more she introduces different bits in the secret shared by Alice and Bob.

Once Alice and Bob generate a shared secret of the desired bit length, they can check for the equality of their secret values without revealing them. For example, if the shared secret is a 64-bit DES key, Alice can send Bob one or more plaintext–ciphertext pairs generated by the DES algorithm using her shared key. Bob also generates the ciphertexts on Alice's plaintexts using his secret key. If the ciphertexts generated by Bob differ from those generated by Alice, Bob becomes confident that their shared

secrets are different and this happened because of the presence of some adversary (or because of communication errors). They then repeat the key-exchange protocol.

Another possible way in which Alice and Bob can gain confidence about the equality of their shared secrets is the use of parity checks. Suppose Alice breaks up her secret in blocks of eight bits and for each block computes the parity bit and sends these bits to Bob. Bob generates the parity bits on the blocks of his secret and compares the two sets of parity bits. If the shared secrets of Alice and Bob differ, it is revealed by this parity check with high probability.

A minor variant of the key-exchange algorithm just described comes with an implementation strategy. The polarization of a photon is measured by an angle θ, $0^\circ \leqslant \theta < 180^\circ$.[1] A photon polarized at an angle θ passes through a ϕ-filter with the probability $\cos^2(\phi - \theta)$ and gets absorbed in the filter with the probability $\sin^2(\phi - \theta)$. Therefore, a photon polarized at the angles $0^\circ, 90^\circ, 45^\circ, 135^\circ$ can be used to represent the quantum states $|0\rangle, |1\rangle, \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle), \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ respectively. Alice and Bob use 0°- and 45°-filters. Alice makes a random choice (x) among the two filters. If $x = 0$, she sends a photon polarized at an angle 0° or 90°. If $x = 1$, a photon polarized at an angle 45° or 135° is sent. When Bob receives the photon transmitted by Alice, he makes a random guess y. If $y = 0$, he uses the 0°-filter to detect its polarization, and if $y = 1$, he uses the 45°-filter to detect its polarization. Then, Alice and Bob reveal their choices x and y and if the two choices agree, they share a common secret bit. See Exercise 8.12 for a mathematical formulation of this strategy.

One of the most startling features of this *Bennett–Brassard algorithm* (often called the *BB84 algorithm*) is that there has been successful experimental implementations of the strategy. The first prototype was designed by the authors themselves in the T. J. Watson Research Center. They used a quantum channel of length 32 cm. Using longer channels requires many technological barriers to be overcome. For example, fiber optic cables tend to weaken and may even destroy the polarization of photons. Using boosters to strengthen the signal is impossible in the quantum mechanical world, since doing so produces an effect similar to eavesdropping. Interference pattern (instead of polarization) has been proposed and utilized to build longer quantum channels for key exchange. At present, Stucki et al. [293] hold the world record of performing quantum key exchange over an (underwater) channel of length 67 km between Geneva and Lausanne.

EXERCISE SET 8.3

8.10 We have exploited the property that $H^2 = I$ in order to prove the correctness of the quantum key-exchange algorithm. Exercise 8.5 lists some other operators (X and Z) which also satisfy the same property ($X^2 = Z^2 = I$). Can one use one of these transforms in place of H in the quantum key-exchange algorithm?

8.11 Assume that Carol eavesdrops (in the manner described in the text) during the execution of the quantum key-exchange protocol between Alice and Bob. Derive for different choices of i, x and z the following probabilities P_{ixz} of having $i \neq j$ in the case $x = y$.

[1] Ask a physicist!

i	x	z	P_{ixz}
0	0	0	0
0	0	1	1/2
0	1	0	1/2
0	1	1	1/2
1	0	0	0
1	0	1	1/2
1	1	0	1/2
1	1	1	1/2

If all these choices of i, x, z are equally likely, show that the probability that Carol introduces mismatch (that is, $i \neq j$) in a shared bit during a random execution of the key-exchange protocol with $x = y$ is $3/8$.
(Note that if $x = y = z = 0$, that is, if the execution of the algorithm proceeds entirely in the classical sense, Carol goes unnoticed. It is the application of the classically meaningless Hadamard transform, that introduces the desired security in the protocol.)

8.12 In the key-exchange algorithm described in the text, Bob (and also Carol) always measure qubits in the classical basis $\{|0\rangle, |1\rangle\}$. Now, consider the following variant of this algorithm. Alice sends, as before, one of the four qubits $|0\rangle, |1\rangle, \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle), \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ depending on her choice of i and x. Bob upon receiving the qubit A generates a random guess $y \in \{0, 1\}$. If $y = 0$, Bob measures A in the classical basis, whereas if $y = 1$, Bob measures A in the basis $\{H|0\rangle, H|1\rangle\}$. After this, they exchange x and y, and retain/discard the bits as in the original algorithm.
(a) Assume that there is no eavesdropping. Argue that this modified strategy works, that is, if $x = y$, we have $i = j$, whereas if $x \neq y$, then $i = j$ with probability $\frac{1}{2}$.
(b) Explain the role of a passive adversary (Carol) in this modified strategy.
(c) Calculate for this variant the probability that Carol introduces an error in a shared bit (when $x = y$).

8.4 Quantum Cryptanalysis

The quantum parallelism has been effectively exploited to design fast (polynomial-time) algorithms to solve some of the intractable mathematical problems discussed in Chapter 4. With the availability of quantum computers, cryptographic systems that derive their security from the intractability of these problems will be unusable (completely insecure). Nobody, however, has the proof that these intractable problems cannot have fast *classical* algorithms. It is interesting to wait and see which (if any) is invented first, a quantum computer or a polynomial-time classical algorithm.

Let us set up some terminology for the rest of this chapter. Let P be a unitary operator on a qubit. One can apply P individually on the i-th bit of an n-bit register. In this case, we denote the operation by P_i. If P_i is operated for each $i = 1, \ldots, n$ (in succession or simultaneously), then we abbreviate $P_1 \cdots P_n$ by the short-hand notation $P^{(n)}$. The parentheses distinguish the operation from P^n which is the n-fold application of P on a *single* qubit.

If P and Q are unitary transforms on n_1- and n_2-bit quantum registers respectively, we let $P \otimes Q$ denote the unitary transform on an $n_1 + n_2$-bit register, with P operating on the left n_1 bits and Q on the right n_2 bits of the register.

8.4.1 Shor's Algorithm for Computing Period

Let $N := 2^n$ for some $n \in \mathbb{N}$. Let $f : \mathbb{Z} \to \mathbb{Z}_N$ be a periodic function with (least) period r, that is, $f(x + kr) = f(x)$ for every $x, k \in \mathbb{Z}$. Suppose further that $1 \ll r \leqslant 2^{n/2}$ and also that $f(0), f(1), \ldots, f(r-1)$ are pairwise distinct. Shor proposed an algorithm for an efficient computation of the period r in this case.

Let's first look at the problem classically. If one evaluates f at randomly chosen points, by the birthday paradox (Exercise 2.172) one requires $\mathrm{O}(\sqrt{r})$ evaluations of f on an average in order to find two different integers x and y with $f(x) = f(y)$. But then $r|(x-y)$. If sufficiently many such pairs (x, y) are available, the period can be obtained by computing the gcd of the integers $x - y$. If r is large, say, $r = \mathrm{O}(2^{n/2})$, this gives us an algorithm for computing r in expected time exponential in n. Shor's quantum algorithm determines r in expected time polynomial in n.

Let us assume that we have an oracle U_f which, on input the $2n$-bit value $|x\rangle_n|y\rangle_n$, computes $|x\rangle_n|f(x) \oplus y\rangle_n$. We prepare a $2n$-bit register A in the state $|0\rangle_n|0\rangle_n$. Then, we apply the Hadamard transform $H^{(n)}$ on the left n-bits. By Exercise 8.8, the state of A becomes

$$\left(H^{(n)} \otimes I^{(n)}\right)|0\rangle_n|0\rangle_n = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle_n|0\rangle_n \,.$$

Supplying this state as the input to the oracle U_f yields the state

$$\frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle_n|f(x)\rangle_n \,.$$

We then measure the output register (right n bits). By the generalized Born rule, we get a value $f(x_0) \in \{0, 1, \ldots, N-1\}$ for some $x_0 \in \{0, 1, \ldots, r-1\}$ and the state of the register A collapses to the uniform superposition of all those $|x\rangle|f(x)\rangle$ for which $f(x) = f(x_0)$. By the given periodicity properties of f, the post-measurement state of the input register (left n bits) can be written as

$$\frac{1}{\sqrt{M}} \sum_{j=0}^{M-1} |x_0 + jr\rangle_n \qquad \textbf{(8.1)}$$

for some M determined by the relations:

$$x_0 + (M-1)r < N \leqslant x_0 + Mr \,.$$

This is an interesting state, for if we were allowed to make copies of this state and measure the different copies, we could collect some values $x_0 + j_1 r, \ldots, x_0 + j_k r$, which in turn would reveal r with high probability. But the no-cloning theorem disallows

making copies of quantum states. Shor proposed a trick to work around with this difficulty. He considered the following transform:

$$F : |x\rangle_n \mapsto \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} e^{2\pi \mathrm{i} xy/N} |y\rangle_n \,. \tag{8.2}$$

By Exercise 8.13, F is a unitary transform. F is known as the *Fourier transform*. Applying F to State (8.1) transforms the input register to the state

$$\begin{aligned} F \frac{1}{\sqrt{M}} \sum_{j=0}^{M-1} |x_0 + jr\rangle_n &= \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} \left(\frac{1}{\sqrt{M}} \sum_{j=0}^{M-1} e^{2\pi \mathrm{i}(x_0+jr)y/N} |y\rangle_n \right) \\ &= \frac{1}{\sqrt{NM}} \sum_{y=0}^{N-1} \left(e^{2\pi \mathrm{i} x_0 y/N} \sum_{j=0}^{M-1} e^{2\pi \mathrm{i} jry/N} \right) |y\rangle_n \,. \end{aligned}$$

A measurement of this state gives an integer $y \in \{0, 1, \ldots, N-1\}$ with the probability

$$p_y := \frac{1}{NM} \left| \sum_{j=0}^{M-1} e^{2\pi \mathrm{i} jry/N} \right|^2$$

Application of the Fourier transform to State (8.1) helps us to concentrate the probabilities of measurement outcomes in strategic states. More precisely, consider a value $y_k = \frac{kN}{r} + \epsilon_k$ of y, where $-1/2 \leqslant \epsilon_k < 1/2$, that is, a value of y close to an integral multiple of N/r. In this case,

$$p_{y_k} = \frac{1}{NM} \left| \sum_{j=0}^{M-1} e^{2\pi \mathrm{i} jr(\frac{kN}{r}+\epsilon_k)/N} \right|^2 = \frac{1}{NM} \left| \sum_{j=0}^{M-1} e^{2\pi \mathrm{i} jr\epsilon_k/N} \right|^2$$

The last summation is that of a geometric series and we have

$$p_{y_k} = \frac{1}{NM} \frac{\sin^2(\pi r \epsilon_k M/N)}{\sin^2(\pi r \epsilon_k/N)} \,.$$

Now, we use the inequalities $\frac{2}{\pi} x \leqslant \sin x \leqslant x$ for $0 \leqslant x \leqslant \pi/2$ and the facts that $rM \approx N$ and that $|\epsilon_k| \leqslant \frac{1}{2}$ to get

$$p_{y_k} \geqslant \frac{1}{NM} \left(\frac{2}{\pi} \frac{\pi r \epsilon_k M}{N} \right)^2 \Bigg/ \left(\frac{\pi r \epsilon_k}{N} \right)^2 = \frac{4}{\pi^2} \frac{M}{N} \approx \frac{4}{\pi^2} \frac{1}{r} \,.$$

Since $\frac{N}{r}$ has about r positive integral multiples less than N and each such multiple has a closest integer y_k for some k, the probability that we obtain one such y_k as the outcome of the measurement is at least $4/\pi^2 = 0.40528\ldots$, that is, after $\mathrm{O}(1)$ iterations of the above procedure we get some y_k. The Fourier transform increases the likelihood of getting some y_k to a level bounded below by a positive constant.

What remains is to show that r can be retrieved from such a *useful* observation y_k. We have $\left| \frac{y_k}{N} - \frac{k}{r} \right| = \left| \frac{\epsilon_k}{N} \right| < \frac{1}{2N} \leqslant \frac{1}{2r^2}$. If a/b and c/d are two distinct rationals

with $b, d \leqslant \sqrt{N}$ and with $\left|\frac{y_k}{N} - \frac{a}{b}\right| < \frac{1}{2N}$ and $\left|\frac{y_k}{N} - \frac{c}{d}\right| < \frac{1}{2N}$, then by the triangle inequality we have $\left|\frac{a}{b} - \frac{c}{d}\right| < \frac{1}{N}$. On the other hand, since $a/b \neq c/d$, we have $\left|\frac{a}{b} - \frac{c}{d}\right| = \left|\frac{ad-bc}{bd}\right| \geqslant \frac{1}{bd} \geqslant \frac{1}{N}$, a contradiction. Therefore, since $r \leqslant \sqrt{N}$, there is a unique rational k/r satisfying $< |\frac{y_k}{N} - \frac{k}{r}| < \frac{1}{2N}$, and this rational k/r can be determined by efficient classical algorithms, for example, using the continued fraction expansion[2] of y_k/N.

If $\gcd(k, r) = 1$, we get r. We can verify this by checking if $f(x) = f(x + r)$. If $\gcd(k, r) > 1$, we get a factor of r. Repeating the entire procedure gives another k'/r, from which we get (hopefully) another factor of r (if not r itself). After a few ($\mathrm{O}(1)$) iterations, we obtain r as the lcm of its factors obtained.

Much of the quantum magic is obtained by the use of the Fourier transform F on a suitably prepared quantum register. The question is then how easy it is to implement F. We will not go to the details, but only mention that a circuit consisting of basic quantum gates and of size $\mathrm{O}(n^2)$ can be used to realize the Fourier transform (cf. Exercise 8.14).

To sum up, we have a polynomial-time (in n) randomized quantum algorithm for computing the period r of f. This leads to efficient quantum algorithms for solving many classically intractable problems of cryptographic significance.

8.4.2 Breaking RSA

Let $m = pq$ with $p, q \in \mathbb{P}$. We have $\phi(m) = (p - 1)(q - 1)$. Choose an RSA key pair (e, d) with $\gcd(e, \phi(m)) = 1$ and $ed \equiv 1 \pmod{\phi(m)}$. Given a message $a \in \mathbb{Z}_m$ the ciphertext message is $b \equiv a^e \pmod{m}$. The task of a cryptanalyst is to compute a from the knowledge of m, e and b. If $\gcd(b, m) > 1$, then this gcd is a non-trivial factor of m. So assume that $b \in \mathbb{Z}_m^*$. But then $a \in \mathbb{Z}_m^*$ also. Since $b \equiv a^e \pmod{m}$, b is in the subgroup of $\mathbb{Z}_m^*$ generated by a. Similarly, $a \equiv b^d \pmod{m}$, that is, a is in the subgroup of $\mathbb{Z}_m^*$ generated by b. It follows that these two subgroups are equal and, in particular, the multiplicative orders of a and b modulo m are the same. This order—call it r—divides $\phi(m)$ and hence is $\leqslant (p - 1)(q - 1) < m$.

Choose $n \in \mathbb{N}$ with $N := 2^n \geqslant m^2 > r^2$. The function $f : \mathbb{Z} \to \mathbb{Z}_N$ sending $x \mapsto b^x \pmod{m}$ is periodic of (least) period r. By Shor's algorithm, one computes r efficiently. Since $\gcd(e, \phi(m)) = 1$ and $r|\phi(m)$, we have $\gcd(e, r) = 1$, that is, using the extended gcd algorithm one obtains an integer d' with $d'e \equiv 1 \pmod{r}$. But then $b^{d'} \equiv a^{d'e} \equiv a \pmod{m}$.

The private key d is the inverse of e modulo $\phi(m)$. It is not necessary to compute d for decrypting b. The inverse d' of e modulo $r = \mathrm{ord}_m(a) = \mathrm{ord}_m(b)$ suffices.

8.4.3 Factoring Integers

Let m be a composite integer that we want to factor. Choose a non-zero integer $a \in \mathbb{Z}_m$. If $\gcd(a, m) > 1$, then we already know a non-trivial factor of m. So assume that $\gcd(a, m) = 1$, that is, $a \in \mathbb{Z}_m^*$. Let $r := \mathrm{ord}_m(a)$.

[2] Consult Zuckerman et al. [316] to learn about continued fractions and their applications in approximating real numbers.

As in the case of breaking RSA, choose $n \in \mathbb{N}$ with $N := 2^n \geqslant m^2 > r^2$. The function $f : \mathbb{Z} \to \mathbb{Z}_N$, $x \mapsto a^x \pmod m$, is periodic of least period r. Shor's algorithm computes r. If r is even, we can write:

$$(a^{r/2} - 1)(a^{r/2} + 1) \equiv 0 \pmod m .$$

Since $\mathrm{ord}_m(a) = r$, $a^{r/2} - 1 \not\equiv 0 \pmod m$. If we also have $a^{r/2} + 1 \not\equiv 0 \pmod m$, then $\gcd(a^{r/2} + 1, m)$ is a non-trivial factor of m. It can be shown that the probability of finding an even r with $a^{r/2} + 1 \not\equiv 0 \pmod m$ is at least half (cf. Exercise 4.9). Thus, trying a few integers $a \in \mathbb{Z}_m^*$ one can factor m.

8.4.4 Computing Discrete Logarithms

A variant of Shor's algorithm in Section 8.4.1 can be used to compute discrete logarithms in the finite field $\mathbb{F}_{p^s}$, $p \in \mathbb{P}$, $s \in \mathbb{N}$. For the sake of simplicity, let us concentrate only on prime fields ($s = 1$). Let g be a generator of $\mathbb{F}_p^* = \mathbb{Z}_p^*$ and our task is to compute for a given $a \in \mathbb{Z}_p^*$ an integer $r \in \{0, 1, \ldots, p-2\}$ with $a \equiv g^r \pmod p$. We assume that p is a large prime, that is, p is odd.

Choose $n \in \mathbb{N}$ with $N := 2^n$ satisfying $p < N < 2p$. We use a $3n$-bit quantum register A in which the left $2n$ bits constitute the input part and the right n bits the output part. The input part is initialized to the uniform superposition of all pairs $(x, y) \in \mathbb{Z}_{p-1}^2$, that is, A has the initial state:

$$\frac{1}{p-1} \sum_{x=0}^{p-2} \sum_{y=0}^{p-2} |x\rangle_n |y\rangle_n |0\rangle_n = \frac{1}{p-1} \sum_{x,y=0}^{p-2} |x\rangle_n |y\rangle_n |0\rangle_n$$

(see Exercise 8.15). Then, we use an oracle

$$U_f : |x\rangle_n |y\rangle_n |z\rangle_n \mapsto |x\rangle_n |y\rangle_n |f(x, y) \oplus z\rangle_n$$

to compute the function $f(x, y) := g^x a^{-y} \pmod p$ in the output register. Applying U_f transforms A to the state

$$\frac{1}{p-1} \sum_{x,y=0}^{p-2} |x\rangle_n |y\rangle_n |g^x a^{-y} \pmod p\rangle_n .$$

Measurement of the output register now gives a value $z \equiv g^k \pmod p$ for some $k \in \{0, 1, \ldots, p-2\}$ and causes the input register to jump to the state

$$\frac{1}{\sqrt{p-1}} \sum_{\substack{x,y=0 \\ x - ry \equiv k \pmod{p-1}}}^{p-2} |x\rangle_n |y\rangle_n .$$

Note that $g^x a^{-y} \equiv g^k \pmod p$ if and only if $x - ry \equiv k \pmod{p-1}$, that is, only those pairs (x, y) that satisfy this congruence contribute to the post-measurement state. For each value of y modulo $p - 1$, we get a unique $x \equiv ry + k \pmod{p-1}$, that is, there are exactly $p - 1$ such pairs (x, y).

If we were allowed make copies of this state and observe two copies separately, we would get pairs (x_1, y_1) and (x_2, y_2) with $x_1 - ry_1 \equiv x_2 - ry_2 \equiv k \pmod{p-1}$. Now, if $\gcd(y_1 - y_2, p-1) = 1$, we would get $r \equiv (y_1 - y_2)^{-1}(x_1 - x_2) \pmod{p-1}$. But we are not allowed to copy quantum states. So Shor used his old trick, that is, applied the Fourier transforms

$$\begin{aligned} |x\rangle_n &\mapsto \frac{1}{\sqrt{N}} \sum_{u=0}^{N-1} e^{2\pi \mathrm{i} xu/N} |u\rangle_n \\ |y\rangle_n &\mapsto \frac{1}{\sqrt{N}} \sum_{v=0}^{N-1} e^{2\pi \mathrm{i} yv/N} |v\rangle_n \end{aligned}$$

to obtain the state

$$\frac{1}{N\sqrt{p-1}} \sum_{u,v=0}^{N-1} \left(\sum_{\substack{x,y=0 \\ x-ry \equiv k \pmod{p-1}}}^{p-2} e^{2\pi \mathrm{i}(xu+yv)/N} \right) |u\rangle_n |v\rangle_n .$$

A measurement of the input register at this state yields $(u, v) \in \{0, 1, \ldots, N-1\}^2$ with probability:

$$p_{u,v} := \frac{1}{N^2(p-1)} \left| \sum_{\substack{x,y=0 \\ x-ry \equiv k \pmod{p-1}}}^{p-2} e^{2\pi \mathrm{i}(xu+yv)/N} \right|^2 . \tag{8.3}$$

As in Shor's period-finding algorithm, we now require to identify a set of *useful* pairs (u, v) which are sufficiently many in number so as to make the probability of observing one of them bounded below by a positive constant. We also need to demonstrate how a useful pair can reveal the unknown discrete logarithm r of a. The jugglery with inequalities and approximations is much more involved in this case. Let us still make a patient attempt to see the end of the story.

First, we eliminate one of x, y from Equation (8.3). Since $x \equiv ry + k \pmod{p-1}$ and $0 \leqslant x \leqslant p-2$, we have $x = (ry + k) \operatorname{rem} (p-1) = (ry + k) - (p-1) \left\lfloor \frac{ry+k}{p-1} \right\rfloor$. But then $xu + yv = \left((ry + k) - (p-1) \left\lfloor \frac{ry+k}{p-1} \right\rfloor \right) u + yv$. Let j be the integer closest to $u(p-1)/N$, that is, $u(p-1) = jN + \epsilon$ with $\epsilon \in \mathbb{Z}$, $-N/2 < \epsilon \leqslant N/2$. This yields

$$xu + yv = -j \left\lfloor \frac{ry + k}{p-1} \right\rfloor N + Sy + T, \tag{8.4}$$

where

$$\begin{aligned} S &:= ru + v - \left(\frac{r}{p-1} \right) \epsilon, \\ T &:= \left(\frac{ry}{p-1} - \left\lfloor \frac{ry+k}{p-1} \right\rfloor \right) \epsilon . \end{aligned} \tag{8.5}$$

Since $j\left\lfloor\frac{ry+k}{p-1}\right\rfloor$ is an integer, substituting Equation (8.4) in Equation (8.3) gives

$$p_{u,v} = \frac{1}{N^2(p-1)}\left|\sum_{y=0}^{p-2} e^{2\pi \mathrm{i} Sy/N} e^{2\pi \mathrm{i} T/N}\right|^2.$$

Writing $S = lN + \sigma$ with $-N/2 < \sigma \leqslant N/2$ then gives

$$p_{u,v} = \frac{1}{N^2(p-1)}\left|\sum_{y=0}^{p-2} e^{2\pi \mathrm{i} \sigma y/N} e^{2\pi \mathrm{i} T/N}\right|^2.$$

We now impose the usefulness conditions on u, v:

$$|\sigma| \leqslant 1/2\,, \tag{8.6}$$
$$|\epsilon| \leqslant N/12\,. \tag{8.7}$$

Involved calculations show that the probability $p_{u,v}$ for a (u, v) satisfying these two conditions is at least $\frac{p-1}{20N^2}$. Let us now see how many pairs (u, v) satisfy the conditions. From Equation (8.5), it follows that for each u there exists a unique v, such that Condition (8.6) is satisfied. Condition (8.7), on the other hand, involves only u. If $w := \mathrm{v}_2(p-1)$, then 2^w must divide ϵ. For each multiple of 2^w not exceeding $N/12$ in absolute value, we get 2^w distinct solutions for u modulo N. (We are solving for u the congruence $u(p-1) \equiv \epsilon \pmod{2^n}$.) There is a total of at least $N/12$ of them. Therefore, the probability of making any one of the useful observations (u, v) is at least $\frac{(p-1)}{20N^2} \times \frac{N}{12} = \frac{(p-1)}{240N} \approx \frac{p}{240N} > \frac{1}{480}$, since $N < 2p$.

We finally explain the extraction of r from a useful observation (u, v). Condition (8.6) and Equation (8.5) give $-\frac{1}{2} < ru + v - \left(\frac{r}{p-1}\right)\epsilon - lN \leqslant \frac{1}{2}$. Dividing throughout by N and using the fact that $u(p-1) = jN + \epsilon$, we get

$$-\frac{1}{2N} < \frac{v}{N} + \frac{rj}{p-1} - l \leqslant \frac{1}{2N},$$

that is, the fractional part of $\frac{v}{N} + \frac{rj}{p-1}$ must lie between $-\frac{1}{2N}$ and $\frac{1}{2N}$. The measurement of the input gives us v and we know N. We approximate $\frac{v}{N}$ to the nearest multiple $\frac{\lambda}{p-1}$ of $\frac{1}{p-1}$ and get $rj \equiv \lambda \pmod{p-1}$. Now, j, being the integer closest to $u(p-1)/N$, is also known to us. If $\gcd(j, p-1) = 1$, we have $r \equiv j^{-1}\lambda \pmod{p-1}$. We don't go into the details of determining the likelihood of the invertibility of j modulo $p-1$. A careful analysis shows that Shor's quantum discrete-log algorithm runs in probabilistic polynomial time (in n).

EXERCISE SET 8.4

8.13 Let F be the Fourier Transform (8.2). For basis vectors $|x\rangle$ and $|x'\rangle$, show that

$$\langle Fx|Fx'\rangle = \begin{cases} 1 & \text{if } x = x', \\ 0 & \text{otherwise.} \end{cases}$$

Conclude that F is a unitary transform.

8.14 Let $N = 2^n$. Let $x, y \in \{0, 1, \ldots, N-1\}$ have binary expansions $(x_{n-1} \ldots x_1 x_0)_2$ and $(y_{n-1} \ldots y_1 y_0)_2$ respectively.
(a) Show that xy/N equals an integer plus the quantity

$$y_{n-1}(.x_0) + y_{n-2}(.x_1 x_0) + y_{n-3}(.x_2 x_1 x_0) + \cdots + y_0(.x_{n-1} x_{n-2} \ldots x_0)\,,$$

where $.x_{k-1} x_{k-2} \ldots x_0 = \frac{x_{k-1}}{2} + \frac{x_{k-2}}{2^2} + \cdots + \frac{x_0}{2^k}$.
(b) Deduce that the quantum Fourier Transform (8.2) can be written as

$$\begin{aligned} F|x\rangle \quad = \quad & \left(\frac{|0\rangle + e^{2\pi \mathrm{i}(.x_0)}|1\rangle}{\sqrt{2}}\right) \otimes \left(\frac{|0\rangle + e^{2\pi \mathrm{i}(.x_1 x_0)}|1\rangle}{\sqrt{2}}\right) \otimes \cdots \otimes \\ & \left(\frac{|0\rangle + e^{2\pi \mathrm{i}(.x_{n-1} x_{n-2} \ldots x_0)}|1\rangle}{\sqrt{2}}\right), \end{aligned}$$

where the i-th expression in parentheses applies to the i-th bit from the left.

8.15 Let $n \in \mathbb{N}$, $N := 2^n$ and $t \in \{1, \ldots, N\}$. Consider an $(n+1)$-bit quantum register with input consisting of the left n bits and the output the rightmost bit. Suppose there is an oracle U_f that takes an n-bit input x and outputs the bit:

$$f(x) := \begin{cases} 0 & \text{if } x < t, \\ 1 & \text{otherwise.} \end{cases}$$

First prepare the register in the state $(H^{(n)} \otimes I)|0\rangle_n |0\rangle_1 = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle_n |0\rangle_1$. Then, apply U_f on this register and finally measure the output bit. Describe the state of the input register after this measurement depending on the outcome of the measurement.

8.16 Recall that the Fourier Transform (8.2) is defined for N equal to a power of 2. It turns out that for such values of N the quantum Fourier transform is *easy* to implement. For this exercise, assume hypothetically that one can efficiently implement F for other values of N too. In particular, take $N = p-1$ in Shor's quantum discrete-log algorithm. Show that in this case, the probability $p_{u,v}$ of Equation (8.3) becomes:

$$p_{u,v} = \frac{1}{(p-1)^3} \left| \sum_{y=0}^{p-2} e^{2\pi \mathrm{i}(ru+v)y/(p-1)} \right|^2 = \begin{cases} \frac{1}{p-1} & \text{if } ru + v \equiv 0 \pmod{p-1}, \\ 0 & \text{otherwise.} \end{cases}$$

Conclude that an outcome (u, v) of measuring the input register yields

$$r \equiv -u^{-1} v \pmod{p-1},$$

provided $\gcd(u, p-1) = 1$.

Chapter Summary

This chapter is a gentle introduction to the recent applications of quantum computation in public-key cryptography. These developments have both good and bad impacts for cryptologers. It is still a big question whether a quantum computer can ever be manufactured. So at present a study of quantum cryptology is mostly theoretical in nature.

Quantum mechanics is governed by a set of four axioms that define a *system* and prescribe the properties of a system. A *quantum bit* (*qubit*) is a quantum mechanical system that has two orthogonal states $|0\rangle$ and $|1\rangle$. A *quantum register* is a collection of qubits of a fixed size.

As an example of what we can gain by using quantum algorithms, we first describe the *Deutsch algorithm* that determines whether a function $f : \{0,1\} \rightarrow \{0,1\}$ is constant by invoking f only once. A classical algorithm requires two invocations.

Next we present the *BB84 algorithm* for key exchange over a quantum mechanical channel. The algorithm guarantees perfect security. This algorithm has been implemented in hardware, and key agreement is carried out over a channel of length 67 km.

Finally, we describe Shor's polynomial-time quantum algorithms for factoring integers and for computing discrete logarithms in finite fields. These algorithms are based on a technique called *quantum Fourier transform*.

If quantum computers can ever be realized, RSA and most other popular cryptosystems described and not described in this book will forfeit all security guarantees. And what will happen to this book? If you don't possess a copy of this wonderful book, just rush to your nearest book store now—they have not yet mastered the quantum technology!

Suggestions for Further Reading

There was a time when the newspapers said that only twelve men understood the theory of relativity. I do not believe there ever was such a time . . . On the other hand, I think I can safely say that nobody understands quantum mechanics.
— Richard Feynman, The Character of Physical Law, BBC, 1965

Quantum mechanics came into existence, when Werner Heisenberg, at the age of 25, proposed the uncertainty principle in 1927. It created an immediate stir in the physics community. Eventually Heisenberg and Niels Bohr came up with an interpretation of quantum mechanics, known as the *Copenhagen interpretation*. While many physicists (like Max Born, Wolfgang Pauli and John von Neumann) subscribed to this interpretation, many other eminent ones (including Albert Einstein, Erwin Schrödinger, Max Planck and Bertrand Russell) did not. Interested readers may consult textbooks by Sakurai [255] and Schiff [258] to study this fascinating area of fundamental science.[3]

For a comprehensive treatment of quantum computation (including cryptographic and cryptanalytic quantum algorithms), we refer the reader to the book by Nielsen and Chuang [218]. Mermin's paper [197] and course notes [198] are also good sources for learning quantum mechanics and computation, and are suitable for computer scientists. Preskill's course notes [244] are also useful, though a bit more physics-oriented. The very readable article [243] by Preskill on the realizability of quantum computers is also worth mentioning in this context. The first known quantum algorithm is due to Deutsch [75].

Bennett and Brassard's quantum key-exchange algorithm (BB84) appeared in [20]. The implementation due to Stucki et al. of this algorithm is reported in [293].

Shor's polynomial-time quantum factorization and discrete-log algorithms are described in [271]. All the details missing in Section 8.4.4 can be found in this paper. No polynomial-time quantum algorithms are known to solve the elliptic curve discrete logarithm problem. Proos and Zalka [245] present an extension of Shor's algorithm for a special class of elliptic curves. See [146] for an adaptation of this algorithm applicable to fields of characteristic 2.

[3] Well! We are not physicists. These books are followed in graduate and advanced undergraduate courses in many institutes and universities.

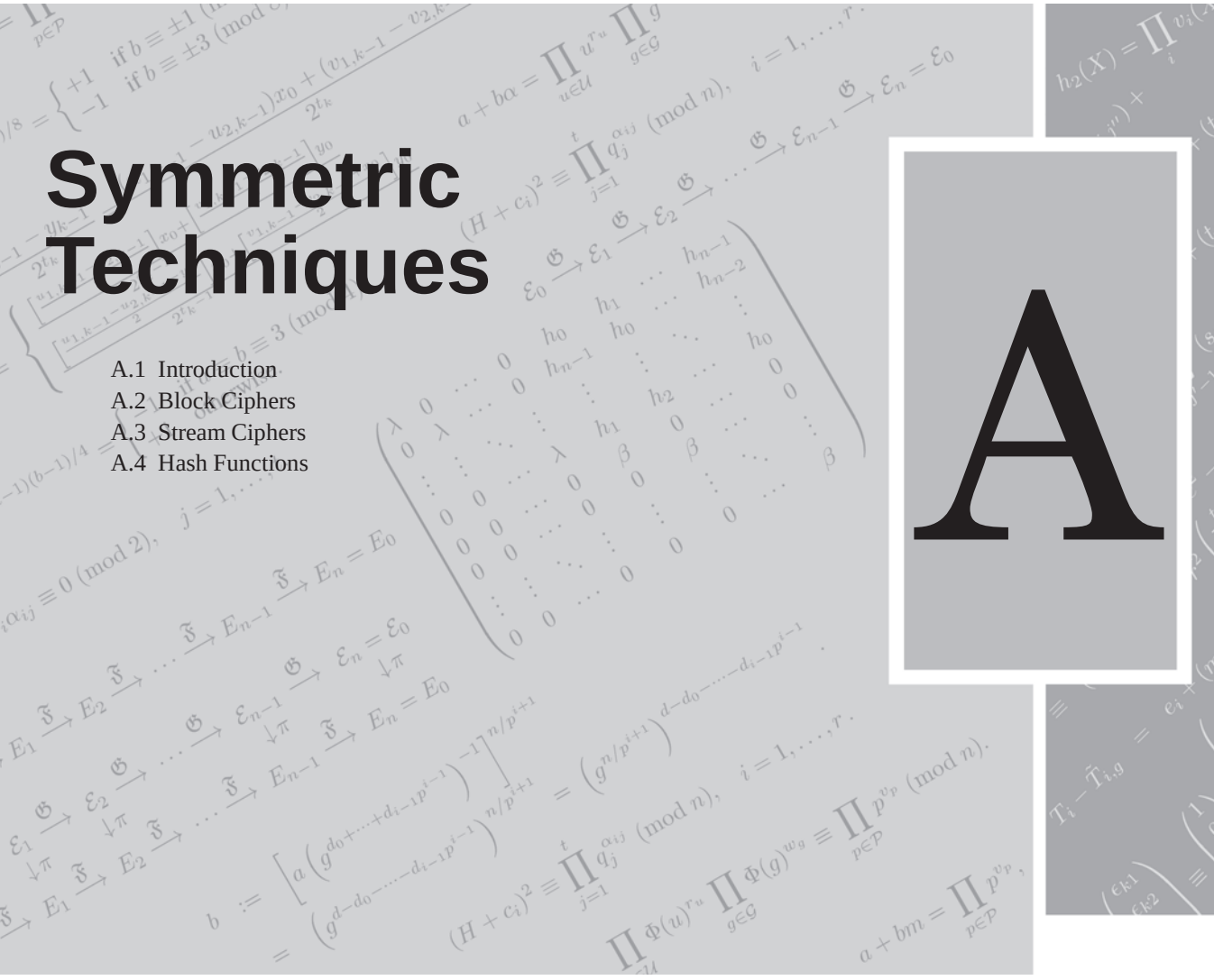
This page is intentionally left blank

Appendices

This page is intentionally left blank

Symmetric Techniques

A.1 Introduction
A.2 Block Ciphers
A.3 Stream Ciphers
A.4 Hash Functions



A

Sour, sweet, bitter, pungent, all must be tasted.
— Chinese Proverb

Unless we change direction, we are likely to end up where we are going.
— Anonymous

Not everything that can be counted counts,
and not everything that counts can be counted.
— Albert Einstein

A.1 Introduction

Cryptography, today, cannot bank solely on public-key (that is, asymmetric) algorithms. Secret-key (that is, symmetric) techniques also have important roles to play. This chapter is an attempt to introduce to the readers some rudimentary notions about symmetric cryptography. The sketchy account that follows lacks both the depth and the breadth of a comprehensive treatment. Given the focus of this book, Appendix A could have been omitted. Nonetheless, some attention to the symmetric technology is never irrelevant for any book on cryptology.

It remains debatable whether hash functions can be treated under the banner of this chapter—a hash function need not even use a key. If the reader is willing to accept *symmetric* as an abbreviation for *not asymmetric*, some justifications can perhaps be given. How does it matter anyway?

A.2 Block Ciphers

Block ciphers encrypt plaintext messages in blocks of fixed lengths and are more ubiquitously used than public-key encryption routines. In a sense, public-key encryption is also block encryption. Since public-key routines are much slower than (secret-key) block ciphers, it is a custom to use public-key algorithms only in specific situations, for example, for encrypting single blocks of data, like keys of symmetric ciphers.

In the rest of this chapter, we use the word *bit* in the conventional sense, that is, to denote a quantity that can take only two possible values, 0 and 1. It is convenient to use the symbol $\mathbb{Z}_2$ to refer to the *set* $\{0, 1\}$. We also let $\mathbb{Z}_2^m$ stand for the set of all bit strings of length m. Whenever we plan to refer to the field (or group) structure of $\mathbb{Z}_2$, we will use the alternative notation $\mathbb{F}_2$.

Definition A.1 A *block cipher* f of block-size n and of key-size r is a map

$$f : \mathbb{Z}_2^n \times \mathbb{Z}_2^r \to \mathbb{Z}_2^n, \quad (m, K) \mapsto c := f(m, K),$$

that encrypts a plaintext block m of bit length n to a ciphertext block c of bit length n under a key K, a bit string of length r. To ensure unique decryption, the map

$$f_K : \mathbb{Z}_2^n \to \mathbb{Z}_2^n, \quad m \mapsto f_K(m) := f(m, K),$$

for a fixed key K has to be a permutation of (that is, a bijective function on) $\mathbb{Z}_2^n$. In that case, the decryption of c to get back m is carried out as $m = f_K^{-1}(c)$. □

A good block cipher has the following desirable properties:

- The sizes n and r should be big enough, so that an adversary cannot exhaustively check all possibilities of m or K in feasible time.
- For most, if not all, keys K, the permutations f_K should be sufficiently random. In other words, if the key K is not known, it should be computationally infeasible to guess the functions f_K and f_K^{-1}. That is, it should be difficult to guess c from

m or m from c, unless the key K is provided. The identity map on $\mathbb{Z}_2^n$, though a permutation of $\mathbb{Z}_2^n$, is a bad candidate for an encryption function f_K. It is also desirable that the functions f_K for different values of K are unpredictably selected from the set of all permutations of $\mathbb{Z}_2^n$. Thus, for example, taking f_K to be a fixed permutation for all choices of K leads to a poor design of a block cipher f.

- For most, if not all, pairs of distinct keys K_1 and K_2, the functions $g_{K_1} \circ g_{K_2}$ should not equal g_K for some key K, where g stands for f or f^{-1} with independent choices in the three uses. A more stringent demand is that the subgroup generated by the permutations f_K for all possible keys K should be a very big subset of the group of all permutations of $\mathbb{Z}_2^n$. If $g_K = g_{K_1} \circ g_{K_2} \circ \cdots \circ g_{K_t}$ for some $t \geqslant 2$, multiple encryption (see Section A.3) forfeits its expected benefits.

A block cipher provably possessing all these good characteristics (in particular, the randomness properties) is difficult to construct in practice. Practical block ciphers are manufactured for reasonably big n and r and come with the hope of representing reasonably unpredictable permutations. We dub a block cipher good or safe, if it stands the test of time. Table A.1 lists some widely used block ciphers.

Table A.1 Some popular block ciphers

Name	n	r
DES (Data Encryption Standard)	64	56
FEAL (Fast Data Encipherment Algorithm)	64	64
SAFER (Secure And Fast Encryption Routine)	64	64
IDEA (International Data Encryption Algorithm)	64	128
Blowfish	64	$\leqslant 448$
Rijndael, accepted as AES (Advanced Encryption Standard) by NIST (National Institute of Standards and Technology, a US government organization)	128/192/256	128/192/256

A.2.1 A Case Study: DES

The *data encryption standard* (*DES*) was proposed as a *federal information processing standard* (*FIPS*) in 1975. DES has been the most popular and the most widely used among all block ciphers ever designed. Although its relatively small key-size offers questionable security under today's computing power, DES still enjoys large-scale deployment in not-so-serious cryptographic applications.

DES encryption requires a 64-bit plaintext block m and a 56-bit key K.[1] Let us plan to use the notations DES_K and DES_K^{-1} to stand respectively for DES encryption and decryption functions under the key K.

[1] A DES key $K = k_1 k_2 \ldots k_{64}$ is actually a 64-bit string. Only 56 bits of K are used for encryption. The remaining 8 bits are used as parity-check bits. Specifically, for each $i = 1, \ldots, 8$ the bit k_{8i} is adjusted so that the i-th byte $(k_{8i-7} k_{8i-6} \ldots k_{8i})$ has an odd number of one-bits.

DES key schedule

The DES algorithm first computes sixteen 48-bit keys $K_1, K_2, \ldots, K_{16}$ from K using a procedure known as the *DES key schedule* described in Algorithm A.1. These 16 keys are used in the 16 rounds of encryption. The key schedule uses two fixed permutations PC1 and PC2 described after Algorithm A.1 and to be read in the row-major order. Here, PC is an abbreviation for *permuted choice*.

Algorithm A.1 **The DES key schedule**

Input: A DES key $K = k_1 k_2 \ldots k_{64}$ (containing the parity-check bits).
Output: Sixteen 48-bit round keys $K_1, K_2, \ldots, K_{16}$.
Steps:
Use PC1 to generate $U_0 := \text{PC1}(K) = k_{57}k_{49}k_{41}\ldots k_{12}k_4 \in \mathbb{Z}_2^{56}$.
Write $U_0 = C_0 \,||\, D_0$ with $C_0, D_0 \in \mathbb{Z}_2^{28}$.
for $i = 1, 2, \ldots, 16$ {
  Take $s := \begin{cases} 1 & \text{if } i = 1, 2, 9, 16, \\ 2 & \text{otherwise.} \end{cases}$
  Cyclically left shift C_{i-1} by s bits to get C_i.
  Cyclically left shift D_{i-1} by s bits to get D_i.
  Let $U_i := C_i \,||\, D_i = u_1 u_2 \ldots u_{56} \in \mathbb{Z}_2^{56}$.
  Compute the i-th round key $K_i := \text{PC2}(U_i) = u_{14}u_{17}u_{11}\ldots u_{29}u_{32}$.
}

PC1

57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

PC2

14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

DES encryption

DES encryption, as described in Algorithm A.2, proceeds in 16 rounds. The i-th round uses the key K_i (obtained from the key schedule) in tandem with the encryption primitive e. A fixed permutation IP and its inverse IP^{-1} are also used.[2]

It requires a specification of the round encryption function e to complete the description of DES encryption. The function e can be compactly depicted as:

$$e(X, J) := \text{P}(S(\text{E}(X) \oplus J)),$$

[2] A block cipher that executes several encryption rounds with the i-th round computing the two halves as $L_i := R_{i-1}$ and $R_i := L_{i-1} \oplus e(R_{i-1}, K_i)$ for some round key K_i and for some encryption primitive e, is called a *Feistel cipher*. Most popular block ciphers mentioned earlier are of this type. Rijndael is an exception, and its acceptance as the new standard has been interpreted as an end of the Feistel dynasty.

Algorithm A.2 **DES encryption**

Input: Plaintext block $m = m_1 m_2 \dots m_{64}$ and the round keys $K_1, \dots, K_{16}$.
Output: The ciphertext block $c = \mathrm{DES}_K(m) \in \mathbb{Z}_2^{64}$.
Steps:
Apply the initial permutation on m to get
$V := \mathrm{IP}(m) = m_{58} m_{50} m_{42} \dots m_{15} m_7 \in \mathbb{Z}_2^{64}$.
Write $V = L_0 \,||\, R_0$ with $L_0, R_0 \in \mathbb{Z}_2^{32}$.
for $i = 1, 2, \dots, 16$ {
/* The i-th encryption round */
$L_i := R_{i-1}$.
$R_i := L_{i-1} \oplus e(R_{i-1}, K_i)$.
}
Let $W := R_{16} \,||\, L_{16} = w_1 w_2 \dots w_{64} \in \mathbb{Z}_2^{64}$.
Apply the inverse of the initial permutation on W to get the ciphertext block
$c := \mathrm{IP}^{-1}(W) = w_{40} w_8 w_{48} \dots w_{57} w_{25} \in \mathbb{Z}_2^{64}$.

IP

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

IP^{-1}

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

where $X \in \mathbb{Z}_2^{32}$, $J \in \mathbb{Z}_2^{48}$, $\mathrm{E} : \mathbb{Z}_2^{32} \to \mathbb{Z}_2^{48}$ is an expansion function, $S : \mathbb{Z}_2^{48} \to \mathbb{Z}_2^{32}$ is a contraction function and P is a fixed permutation of $\mathbb{Z}_2^{32}$ (called the permutation function). S uses eight S-boxes (substitution boxes) $S_1, S_2, \dots, S_8$. Each S-box S_j is a 4×16 matrix with each row a permutation of $0, 1, 2, \dots, 15$ and is used to convert a 6-bit string $y_1 y_2 y_3 y_4 y_5 y_6$ to a 4-bit string $z_1 z_2 z_3 z_4$ as follows. Let μ denote the integer with binary representation $y_1 y_6$ and ν the integer with binary representation $y_2 y_3 y_4 y_5$. Then, $z_1 z_2 z_3 z_4$ is the 4-bit binary representation of the μ, ν-th entry in the matrix S_j. (Here, the numbering of the rows and columns starts from 0.) In this case, we write $S_j(y_1 y_2 y_3 y_4 y_5 y_6) = z_1 z_2 z_3 z_4$. Algorithm A.3 provides the description of e.

Algorithm A.3 **The DES round encryption primitive e**

Input: $X \in \mathbb{Z}_2^{32}$ and $J \in \mathbb{Z}_2^{48}$.
Output: $e(X, J)$.
Steps:
$Y := \mathrm{E}(X) \oplus J$ (where $\mathrm{E}(x_1 x_2 \dots x_{32}) = x_{32} x_1 x_2 \dots x_{32} x_1$).
Write $Y = Y_1 \,||\, Y_2 \,||\, \cdots \,||\, Y_8$ with each $Y_j \in \mathbb{Z}_2^6$.
for $j = 1, 2, \dots, 8$ { $Z_j := S_j(Y_j) \in \mathbb{Z}_2^4$. }
$Z := Z_1 \,||\, Z_2 \,||\, \cdots \,||\, Z_8 \in \mathbb{Z}_2^{32}$.
$e(X, J) := \mathrm{P}(Z) \in \mathbb{Z}_2^{32}$ (where $\mathrm{P}(z_1 z_2 \dots z_{32}) = z_{16} z_7 z_{20} \dots z_4 z_{25}$).

The tables for E and P are as follows.

E

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

P

16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

Finally, the eight S-boxes are presented:

S_1

14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

S_2

15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

S_3

10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

S_4

7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

S_5

2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

S_6

12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

S_7

4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

S_8

13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

DES decryption

DES decryption is analogous to DES encryption. To obtain $m := \mathrm{DES}_K^{-1}(c)$ one first computes the round keys $K_1, K_2, \ldots, K_{16}$ using Algorithm A.1. One then calls a minor variant of Algorithm A.2. First, the roles of m and c are interchanged. That is, one inputs c instead of m, and obtains m in place of c as output. Moreover, the right half R_i in the i-th round is computed as $R_i := L_{i-1} \oplus e(R_{i-1}, K_{17-i})$. In other words, DES decryption is same as DES encryption, only with the sequence of using the keys $K_1, K_2, \ldots, K_{16}$ reversed. Solve Exercise A.1 in order to establish the correctness of this decryption procedure.

DES test vectors

Some test vectors for DES are given in Table A.2.

Table A.2 DES test vectors

Key	Plaintext block	Ciphertext block
0101010101010101	0000000000000000	8ca64de9c1b123a7
fefefefefefefefe	ffffffffffffffff	7359b2163e4edc58
3101010101010101	1000000000000001	958e6e627a05557B
1010101010101010	1111111111111111	f40379ab9e0ec533
0123456789abcdef	1111111111111111	17668dfc7292532d
1010101010101010	0123456789abcdef	8a5ae1f81ab8f2dd
fedcba9876543210	0123456789abcdef	ed39d950fa74bcc4

Cryptanalysis of DES

DES, being a popular block cipher, has gone through a good amount of cryptanalytic studies. At present, *linear cryptanalysis* and *differential cryptanalysis* are the most sophisticated attacks on DES. But the biggest problem with DES is its relatively small key size (56 bits). An exhaustive key search for a given plaintext–ciphertext pair needs carrying out a maximum of 2^{56} encryptions in order to obtain the correct key. But how big is this number $2^{56} = 72{,}057{,}594{,}037{,}927{,}936$ (nearly 72 quadrillion) in a cryptographic sense?

In order to review this question, RSA Security Inc. posed several challenges for obtaining the DES key from given plaintext–ciphertext pairs. The first challenge, posed in January 1997, was broken by Rocke Verser of Loveland, Colorado, with approximately 96 days of computing. DES Challenge II-1 was broken in February 1998 by

distributed.net with 41 days of computing, and the DES challenge II-2 was cracked in July 1998 by the Electronic Frontier Foundation (EFF) in just 56 hours. Finally, DES Challenge III was broken in a record of 22 hours 15 minutes in January 1999. The computations were carried out in EFF's supercomputer *Deep Crack* with collaborative efforts from nearly 10^5 PCs on the Internet guided by distributed.net. These figures demonstrate that DES offers hardly any security against a motivated adversary.

Another problem with DES is that its design criteria (most importantly, the objectives behind choosing the particular S-boxes) were never made public. Chances remain that there are hidden backdoors, though none has been discovered till date.

A.2.2 The Advanced Standard: AES

The *advanced encryption standard* (*AES*) [219] has superseded the older standard DES. The Rijndael cipher designed by Daemen and Rijmen has been accepted as the advanced standard. As mentioned in Footnote 2, Rijndael is *not* a Feistel cipher. Its working is based on the arithmetic in the finite field $\mathbb{F}_{2^8} = \mathbb{F}_{256}$ and in the finite ring $A := \mathbb{F}_{2^8}[Y]/\langle Y^4 + 1\rangle$.

Data representation

AES encrypts data in blocks of 128 bits. Let $B = b_0 b_1 \dots b_{127}$ be a block of data, where each b_i is a bit. Keeping in view typical 32-bit processors, each such block B is represented as a sequence of four 32-bit words, that is, $B = B_0 B_1 B_2 B_3$, where B_i represents the bit string $b_{32i} b_{32i+1} \dots b_{32i+31}$. Each word $C = c_0 c_1 \dots c_{31}$, in turn, is viewed as a sequence of four octets, that is, $C = C_0 C_1 C_2 C_3$, where C_i stores the bit string $c_{8i} c_{8i+1} \dots c_{8i+7}$. Each octet is identified as an element of $\mathbb{F}_{2^8}$, whereas an entire 32-bit word is identified with an element of $\mathbb{F}_{2^8}[Y]/\langle Y^4 + 1\rangle$.

The field $\mathbb{F}_{2^8}$ is represented as $\mathbb{F}_2[X]/\langle f(X)\rangle$, where $f(X)$ is the irreducible polynomial $X^8 + X^4 + X^3 + X + 1$. Let $x := X + \langle f(X)\rangle$. The element $d_7 x^7 + d_6 x^6 + \dots + d_1 x + d_0 \in \mathbb{F}_{2^8}$ is identified with the octet $d_7 d_6 \dots d_1 d_0$. Thus, the i-th octet $c_{8i} c_{8i+1} \dots c_{8i+7}$ in a word is treated as the finite field element $c_{8i} x^7 + c_{8i+1} x^6 + \dots + c_{8i+6} x + c_{8i+7} \in \mathbb{F}_{2^8}$.

Now, let us explain the interpretation of a 32-bit word $C = C_0 C_1 C_2 C_3$. The $\mathbb{F}_{2^8}$-algebra $A := \mathbb{F}_{2^8}[Y]/\langle Y^4 + 1\rangle$ is not a field, since the polynomial $Y^4 + 1$ is reducible (over $\mathbb{F}_2$ and so over $\mathbb{F}_{2^8}$). However, each element β of A can be uniquely expressed as a polynomial $\beta = \alpha_3 y^3 + \alpha_2 y^2 + \alpha_1 y + \alpha_0$, where $y := Y + \langle Y^4 + 1\rangle$ and where each α_i is an element of $\mathbb{F}_{2^8}$. As described in the last paragraph, each α_i is represented as an octet. We take C_i to be the octet representing α_{3-i}, that is, the 32-bit word $\alpha_3 \alpha_2 \alpha_1 \alpha_0$ stands for the element $\beta \in A$.

$\mathbb{F}_{2^8}$ and A are rings and hence equipped with arithmetic operations (addition and multiplication). These operations are different from the usual addition and multiplication operations defined on octets and words. For example, the addition of two octets or words under the AES interpretation is the same as bit-wise XOR of octets or words. The AES multiplication of octets and words, on the other hand, involves polynomial arithmetic and reduction modulo the defining polynomials and so cannot be expressed so simply as addition. To resolve ambiguities, let us plan to denote the multiplication

of $\mathbb{F}_{2^8}$ by $\odot$ and that of A by $\otimes$, whereas regular multiplication symbols ($\cdot$, $\times$ and juxtaposition) stand for the standard multiplication on octets or words. Exercises A.5, A.6 and A.7 discuss about efficient implementations of the arithmetic in $\mathbb{F}_{2^8}$ and A.

Every non-zero element $\alpha \in \mathbb{F}_{2^8}$ is invertible; the inverse is denoted by α^{-1} and can be computed by the extended gcd algorithm on polynomials over $\mathbb{F}_2$. With an abuse of notation, we take $0^{-1} := 0$. Every non-zero element of A is not invertible (under the multiplication of A). The AES algorithm uses the following invertible element $\beta := 03010102$ (in hex notation); its inverse is $\beta^{-1} = 0\text{b}0\text{d}090\text{e}$.

The AES algorithm uses an object called a *state*, comprising 16 octets arranged in a 4×4 array. Each message block also consists of 16 octets. Let $M = \mu_0\mu_1 \ldots \mu_{15}$ be a message block (of 16 octets). This block is translated to a state as follows:

$$\mu_0\mu_1 \ldots \mu_{15} \mapsto \begin{array}{|c|c|c|c|}\hline \mu_0 & \mu_4 & \mu_8 & \mu_{12} \\ \hline \mu_1 & \mu_5 & \mu_9 & \mu_{13} \\ \hline \mu_2 & \mu_6 & \mu_{10} & \mu_{14} \\ \hline \mu_3 & \mu_7 & \mu_{11} & \mu_{15} \\ \hline \end{array} . \qquad \textbf{(A.1)}$$

Thus, each word in the block is relocated in a column of the state. At the end of the encryption procedure, AES makes the reverse translation of a state to a block:

$$\begin{array}{|c|c|c|c|}\hline \sigma_{00} & \sigma_{01} & \sigma_{02} & \sigma_{03} \\ \hline \sigma_{10} & \sigma_{11} & \sigma_{12} & \sigma_{13} \\ \hline \sigma_{20} & \sigma_{21} & \sigma_{22} & \sigma_{23} \\ \hline \sigma_{30} & \sigma_{31} & \sigma_{32} & \sigma_{33} \\ \hline \end{array} \mapsto \sigma_{00}\sigma_{10}\sigma_{20}\sigma_{30}\sigma_{01}\sigma_{11} \ldots \sigma_{23}\sigma_{33} . \qquad \textbf{(A.2)}$$

AES key schedule

A collection of round keys is generated from the given AES key K. The number of rounds of the AES encryption algorithm depends on the size of the key. Let us denote the number of words in the AES key by N_k and the corresponding number of rounds by N_r. We have:

$$N_r = \begin{cases} 10 & \text{if } N_k = 4 \text{ (that is, for AES-128),} \\ 12 & \text{if } N_k = 6 \text{ (that is, for AES-192),} \\ 14 & \text{if } N_k = 8 \text{ (that is, for AES-256).} \end{cases}$$

One first generates an initial 128-bit key $K_0K_1K_2K_3$. Subsequently, for the i-th round, $1 \leqslant i \leqslant N_r$, a 128-bit key $K_{4i}K_{4i+1}K_{4i+2}K_{4i+3}$ is required. Here, each K_j is a 32-bit word. The key schedule (also called *key expansion*) generates a total of $4(N_r + 1)$ words $K_0, K_1, \ldots, K_{4N_r+3}$ from the given secret key K using a procedure described in Algorithm A.4. Here, $(02)^{j-1}$ stands for the octet that represents the element $x^{j-1} \in \mathbb{F}_{2^8}$. The following table summarizes these values for $j = 1, 2, \ldots, 15$.

j	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
x^{j-1}	01	02	04	08	10	20	40	80	1b	36	6c	d8	ab	4d	9a

The transformation $\mathrm{SubWord}$ on a word $T = \tau_0\tau_1\tau_2\tau_3$ is the octet-wise application of AES S-box substitution $\mathrm{SubOctet}$, that is,

$$\mathrm{SubWord}(T) = \mathrm{SubOctet}(\tau_0) \,||\, \mathrm{SubOctet}(\tau_1) \,||\, \mathrm{SubOctet}(\tau_2) \,||\, \mathrm{SubOctet}(\tau_3).$$

Algorithm A.4 **AES key schedule**

Input: (N_k and) the secret key $K = \kappa_0\kappa_1 \ldots \kappa_{4N_k-1}$, where each κ_i is an octet.
Output: The expanded keys $K_0, K_1, \ldots, K_{4N_r+3}$.
Steps:
/* Initially copy the bytes of K */
for $i = 0, 1, \ldots, N_k - 1$ { $K_i := \kappa_{4i}\kappa_{4i+1}\kappa_{4i+2}\kappa_{4i+3}$. }

/* Recursively define the round keys */
for $i = N_k, N_k + 1, \ldots, 4N_r + 3$ {
    $T := K_{i-1}$; /* T is a temporary word variable. */
    /* Let $T = \tau_0\tau_1\tau_2\tau_3$, where each τ_i is an octet. */
    if ($i \text{ rem } N_k = 0$) { $T := \mathrm{SubWord}(\tau_1\tau_2\tau_3\tau_0) \oplus [(02)^{(i/N_k)-1}||000000]$. }
    else if ($N_k > 6$) and ($i \text{ rem } N_k = 4$) { $T := \mathrm{SubWord}(T)$. }
    $K_i := K_{i-N_k} \oplus T$.
}

The transformation $\mathrm{SubOctet}$ is also used in each encryption round and is now described. Let $A = a_0a_1 \ldots a_7$ be an octet that can be identified with an element of $\mathbb{F}_{2^8}$ as mentioned earlier. Let $B = b_0b_1 \ldots b_7$ denote the octet representing the inverse of this finite field element. (We take $0^{-1} = 0$.) One then applies the following affine transformation on B to generate the final value $C := \mathrm{SubOctet(A)} := c_0c_1 \ldots c_7$. Here, $D = d_0d_1 \ldots d_7$ is the constant octet $63 = 01100011$.

$$c_i := b_i \oplus b_{(i+1)\mathrm{rem}8} \oplus b_{(i+2)\mathrm{rem}8} \oplus b_{(i+3)\mathrm{rem}8} \oplus b_{(i+4)\mathrm{rem}8} \oplus d_i \,. \qquad \textbf{(A.3)}$$

In order to speed up this octet substitution, one may use table lookup. Since the output octet C depends only on the input octet A, one can precompute a table of values of $\mathrm{SubOctet}(A)$ for the 256 possible values of A. This list is given in Table A.3. The table is to be read in the row-major fashion. In other words, if *hi* and *lo* respectively represent the most and the least significant four bits of A, then $\mathrm{SubOctet}(A)$ can be read off from the entry in the table having row number *hi* and column number *lo*. For example, $\mathrm{SubOctet(a7)} = 5\mathrm{c}$. In an actual implementation, a one-dimensional array is to be used. We use a two-dimensional format in Table A.3 for the sake of clarity of presentation.

AES encryption

AES encryption is described in Algorithm A.5. The algorithm first converts the input plaintext message block to a state, applies a series of transformations on this state and finally converts the state back to a message (the ciphertext).

Table A.3 AES S-box

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

The individual state transition transformations are now explained. The transition $\mathrm{SubState}$ is an octet-by-octet application of the substitution function $\mathrm{SubOctet}$, that is, $\mathrm{SubState}$ maps

$$\begin{array}{|c|c|c|c|}\hline \sigma_{00} & \sigma_{01} & \sigma_{02} & \sigma_{03} \\ \hline \sigma_{10} & \sigma_{11} & \sigma_{12} & \sigma_{13} \\ \hline \sigma_{20} & \sigma_{21} & \sigma_{22} & \sigma_{23} \\ \hline \sigma_{30} & \sigma_{31} & \sigma_{32} & \sigma_{33} \\ \hline \end{array} \mapsto \begin{array}{|c|c|c|c|}\hline \sigma'_{00} & \sigma'_{01} & \sigma'_{02} & \sigma'_{03} \\ \hline \sigma'_{10} & \sigma'_{11} & \sigma'_{12} & \sigma'_{13} \\ \hline \sigma'_{20} & \sigma'_{21} & \sigma'_{22} & \sigma'_{23} \\ \hline \sigma'_{30} & \sigma'_{31} & \sigma'_{32} & \sigma'_{33} \\ \hline \end{array},$$

where $\sigma'_{rc} = \mathrm{SubOctet}(\sigma_{rc})$ for all r, c. The transform $\mathrm{ShiftRows}$ cyclically left rotates the r-th row by r byte positions, that is, maps

$$\begin{array}{|c|c|c|c|}\hline \sigma_{00} & \sigma_{01} & \sigma_{02} & \sigma_{03} \\ \hline \sigma_{10} & \sigma_{11} & \sigma_{12} & \sigma_{13} \\ \hline \sigma_{20} & \sigma_{21} & \sigma_{22} & \sigma_{23} \\ \hline \sigma_{30} & \sigma_{31} & \sigma_{32} & \sigma_{33} \\ \hline \end{array} \mapsto \begin{array}{|c|c|c|c|}\hline \sigma_{00} & \sigma_{01} & \sigma_{02} & \sigma_{03} \\ \hline \sigma_{11} & \sigma_{12} & \sigma_{13} & \sigma_{10} \\ \hline \sigma_{22} & \sigma_{23} & \sigma_{20} & \sigma_{21} \\ \hline \sigma_{33} & \sigma_{30} & \sigma_{31} & \sigma_{32} \\ \hline \end{array}.$$

The AddKey operation uses four 32-bit round keys L_0, L_1, L_2, L_3. Name the octets of L_i as $\lambda_{i0}\lambda_{i1}\lambda_{i2}\lambda_{i3}$. The i-th key L_i is XORed with the i-th column of the state, that is, AddKey transforms

$$\begin{array}{|c|c|c|c|}\hline \sigma_{00} & \sigma_{01} & \sigma_{02} & \sigma_{03} \\ \hline \sigma_{10} & \sigma_{11} & \sigma_{12} & \sigma_{13} \\ \hline \sigma_{20} & \sigma_{21} & \sigma_{22} & \sigma_{23} \\ \hline \sigma_{30} & \sigma_{31} & \sigma_{32} & \sigma_{33} \\ \hline \end{array} \mapsto \begin{array}{|c|c|c|c|}\hline \sigma_{00} \oplus \lambda_{00} & \sigma_{01} \oplus \lambda_{10} & \sigma_{02} \oplus \lambda_{20} & \sigma_{03} \oplus \lambda_{30} \\ \hline \sigma_{10} \oplus \lambda_{01} & \sigma_{11} \oplus \lambda_{11} & \sigma_{12} \oplus \lambda_{21} & \sigma_{13} \oplus \lambda_{31} \\ \hline \sigma_{20} \oplus \lambda_{02} & \sigma_{21} \oplus \lambda_{12} & \sigma_{22} \oplus \lambda_{22} & \sigma_{23} \oplus \lambda_{32} \\ \hline \sigma_{30} \oplus \lambda_{03} & \sigma_{31} \oplus \lambda_{13} & \sigma_{32} \oplus \lambda_{23} & \sigma_{33} \oplus \lambda_{33} \\ \hline \end{array}.$$

Finally, the $\mathrm{MixCols}$ transform multiplies each column of the state, regarded as an element of $A = \mathbb{F}_{2^8}[Y]/\langle Y^4+1\rangle$, by the element $[03]y^3+[01]y^2+[01]y+[02] \in A$, where

Algorithm A.5 **AES encryption**

Input: The plaintext message $M = \mu_0\mu_1 \dots \mu_{15}$ and the round keys $K_0, K_1, \dots, K_{4N_r+3}$.
Output: Ciphertext message $C = \gamma_0\gamma_1 \dots \gamma_{15}$.
Steps:
Convert M to the state S. /* Use Transform (A.1) */
$S := \mathrm{AddKey}(S, K_0, K_1, K_2, K_3)$.
for $i = 1, 2, \dots, N_r$ {
    $S := \mathrm{SubState}(S)$.
    $S := \mathrm{ShiftRows}(S)$.
    if ($i \neq N_r$) { $S := \mathrm{MixCols}(S)$. }
    $S := \mathrm{AddKey}(S, K_{4i}, K_{4i+1}, K_{4i+2}, K_{4i+3})$.
}
Convert S to the message C. /* Use Transform (A.2) */

the coefficients (expressions within square brackets) are octet values in hexadecimal, that can be identified with elements of $\mathbb{F}_{2^8}$. For the c-th column, this transformation can be represented as:

$$\begin{pmatrix} \sigma_{0c} \\ \sigma_{1c} \\ \sigma_{2c} \\ \sigma_{3c} \end{pmatrix} \mapsto \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \odot \begin{pmatrix} \sigma_{0c} \\ \sigma_{1c} \\ \sigma_{2c} \\ \sigma_{3c} \end{pmatrix}.$$

AES decryption

AES decryption involves taking inverse of each state transition performed during encryption. The key schedule needed for encryption is to be used during decryption too. The straightforward decryption routine is given in Algorithm A.6.

Algorithm A.6 **AES decryption**

Input: The ciphertext message $C = \gamma_0\gamma_1 \dots \gamma_{15}$ and the round keys $K_0, K_1, \dots, K_{4N_r+3}$.
Output: The recovered plaintext message $M = \mu_0\mu_1 \dots \mu_{15}$.
Steps:
Convert C to the state S. /* Use Transform (A.1) */
$S := \mathrm{AddKey}(S, K_{4N_r}, K_{4N_r+1}, K_{4N_r+2}, K_{4N_r+3})$.
for $i = N_r - 1, N_r - 2, \dots, 1, 0$ {
    $S := \mathrm{ShiftRows}^{-1}(S)$.
    $S := \mathrm{SubState}^{-1}(S)$.
    $S := \mathrm{AddKey}(S, K_{4i}, K_{4i+1}, K_{4i+2}, K_{4i+3})$.
    if ($i \neq 0$) { $S := \mathrm{MixCols}^{-1}(S)$. }
}
Convert S to the message M. /* Use Transform (A.2) */

Table A.4 Inverse of AES S-box

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

What remains is a description of the inverses of the basic state transformations. AddKey involves octet-by-octet XORing and so is its own inverse. Table A.4 summarizes the inverse of the substitution transition $\mathrm{SubOctet}$ (Exercise A.8). For computing $\mathrm{SubState}^{-1}(S)$, one should apply $\mathrm{SubOctet}^{-1}$ on each octet of S. The inverse of $\mathrm{ShiftRows}$ is also straightforward and can be given by

$$\begin{array}{|c|c|c|c|} \hline \sigma_{00} & \sigma_{01} & \sigma_{02} & \sigma_{03} \\ \hline \sigma_{10} & \sigma_{11} & \sigma_{12} & \sigma_{13} \\ \hline \sigma_{20} & \sigma_{21} & \sigma_{22} & \sigma_{23} \\ \hline \sigma_{30} & \sigma_{31} & \sigma_{32} & \sigma_{33} \\ \hline \end{array} \mapsto \begin{array}{|c|c|c|c|} \hline \sigma_{00} & \sigma_{01} & \sigma_{02} & \sigma_{03} \\ \hline \sigma_{13} & \sigma_{10} & \sigma_{11} & \sigma_{12} \\ \hline \sigma_{22} & \sigma_{23} & \sigma_{20} & \sigma_{21} \\ \hline \sigma_{31} & \sigma_{32} & \sigma_{33} & \sigma_{30} \\ \hline \end{array} .$$

Finally, $\mathrm{MixCols}^{-1}$ involves multiplication of each column by the inverse of the element $[03]y^3 + [01]y^2 + [01]y + [02] \in A$, that is, by the element $[0\mathrm{b}]y^3 + [0\mathrm{d}]y^2 + [09]y + [0\mathrm{e}]$. So $\mathrm{MixCols}^{-1}$ transforms each column of the state as follows:

$$\begin{pmatrix} \sigma_{0c} \\ \sigma_{1c} \\ \sigma_{2c} \\ \sigma_{3c} \end{pmatrix} \mapsto \begin{pmatrix} 0\mathrm{e} & 0\mathrm{b} & 0\mathrm{d} & 09 \\ 09 & 0\mathrm{e} & 0\mathrm{b} & 0\mathrm{d} \\ 0\mathrm{d} & 09 & 0\mathrm{e} & 0\mathrm{b} \\ 0\mathrm{b} & 0\mathrm{d} & 09 & 0\mathrm{e} \end{pmatrix} \odot \begin{pmatrix} \sigma_{0c} \\ \sigma_{1c} \\ \sigma_{2c} \\ \sigma_{3c} \end{pmatrix} .$$

AES decryption is as efficient as AES encryption, since each state transformation primitive has the same structure as its inverse. However, the sequence of application of these primitives in the loop (rounds) for decryption differs from that for encryption. For some implementations, mostly in hardware, this may be a problem. Compare this with DES for which the encryption and decryption algorithms are identical save the sequence of using the round keys (Exercise A.1). With little additional effort AES can also be furnished with this useful property of DES. All we have to do is to use a different key schedule for decryption. The necessary modifications are explored in Exercise A.9.

AES test vectors

Table A.5 provides the ciphertexts for the plaintext block

$$M = \texttt{00112233445566778899aabbccddeeff}$$

under different keys.

Table A.5 AES test vectors

Cipher	Key	Ciphertext block
AES-128	`0001020304050607 \ 08090a0b0c0d0e0f`	`69c4e0d86a7b0430 \ d8cdb78070b4c55a`
AES-192	`0001020304050607 \ 08090a0b0c0d0e0f \ 1011121314151617`	`dda97ca4864cdfe0 \ 6eaf70a0ec0d7191`
AES-256	`0001020304050607 \ 08090a0b0c0d0e0f \ 1011121314151617 \ 18191a1b1c1d1e1f`	`8ea2b7ca516745bf \ eafc49904b496089`

Cryptanalysis of AES

AES has been designed so that *linear* and *differential attacks* are infeasible. Another attack known as the *square attack* has been proposed by Lucks [184] and Ferguson et al. [93], but at present can tackle less number of rounds than used in Rijndael encryption. Also see Gilbert and Minier [112] to know about the *collision attack*.

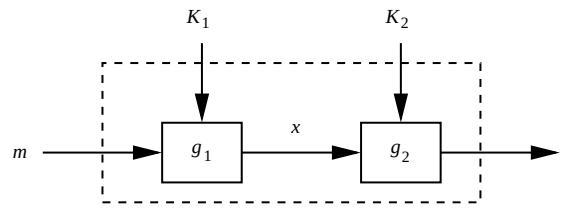
The distinct algebraic structure of AES encryption invites special algebraic attacks. One such potential attack (the *XSL attack*) has been proposed by Courtois and Pieprzyk [68]. Although this attack has not yet been proved to be effective, a better understanding of the algebra may, in foreseeable future, lead to disturbing consequences for the advanced standard.

For more information on AES, read the book [71] from the designers of the cipher. Also visit the following Internet sites:

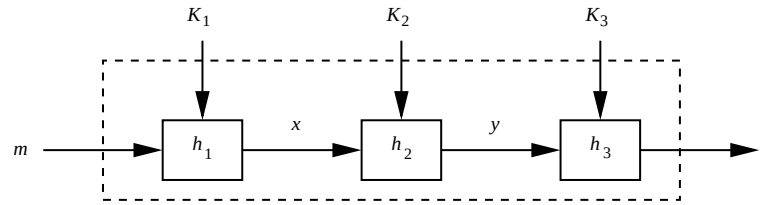
http://www.esat.kuleuven.ac.be/∼rijmen/rijndael/	Rijndael home
http://csrc.nist.gov/CryptoToolkit/aes/index1.html	NIST site for AES
http://www.cryptosystem.net/aes/	Algebraic attacks

A.2.3 Multiple Encryption

Multiple encryption presents a way to achieve a desired level of security by using block ciphers of small key sizes. The idea is to cascade several stages of encryption and/or decryption, with different stages working under different keys. Figure A.1 illustrates double and triple encryption for a block cipher f. Each g_i or h_j represents either the encryption or the decryption function of f under the given key.



(a) Double encryption



(b) Triple encryption

Figure A.1 Multiple encryption

For double encryption, we have $K_1 \neq K_2$ and both g_1 and g_2 are usually the encryption function. Unless $f_{K_2} \circ f_{K_1}$ is the same as f_K for some key K and if the permutations of f are reasonably random, it appears at the first glance that double encryption increases the effective key size by a factor of two. Unfortunately, this is not the case. The *meet-in-the-middle attack* on double encryption works as follows.

Suppose that an adversary knows a plaintext–ciphertext pair (m, c) under the unknown keys K_1, K_2. We assume as before that f has block-size n and key-size r. The adversary computes for each possibility of $K_1 = i \in \{0, 1, \ldots, 2^r - 1\}$ the encrypted message $x_i := f_i(m)$. She also computes for each $K_2 = j \in \{0, 1, \ldots, 2^r - 1\}$ the decrypted message $x'_j := f_j^{-1}(c)$. Now, (i, j) is a possible value of (K_1, K_2) if and only if $x_i = x'_j$.

A given pair (m, c) usually gives many such candidates (i, j) for (K_1, K_2). More precisely, if each f_j^{-1} is assumed to be a random permutation of $\mathbb{Z}_2^n$, for a given i we have the equality $x_i = x'_j$ for an expected number of $2^r/2^n$ values of j. Considering all possibilities for i gives an expected number of $2^r \times 2^r/2^n = 2^{2r-n}$ candidate pairs (i, j). If $f = \mathrm{DES}$, this number is $2^{2\times 56-64} = 2^{48}$.

If a second pair (m', c') under (K_1, K_2) is also known to the adversary, then for a given i the pair (i, j) is consistent with both (m, c) and (m', c') with probability $2^r/(2^n \times 2^n)$. Thus, we get an expected number of $(2^r \times 2^r)/(2^n \times 2^n) = 2^{2r-2n}$ candidates (i, j). For DES, this number is 2^{-16}. This implies that it is very unlikely that a false candidate (i, j) satisfies both (m, c) and (m', c'). Thus, with high probability the adversary uniquely identifies the double DES key (K_1, K_2) from two plaintext–ciphertext pairs.

This attack calls for $\mathrm{O}(2^r)$ encryptions and $\mathrm{O}(2^r)$ decryptions. With the assumption that each encryption takes roughly the same time as each decryption (as in the case of DES), the adversary spends a time for $\mathrm{O}(2^r)$ encryptions. Moreover, she can find all

the matches $x_i = x'_j$ in $O(r2^r)$ time. This implies that double encryption increases the effective key size (over single encryption) by a few bits only. On the other hand, both the actual key size and the encryption time get doubled. In view of these shortcomings, double encryption is rarely used in practice.

For the triple encryption scheme of Figure A.1, a meet-in-the-middle attack at x or y demands an effort equivalent to $O(2^{2r})$ encryptions, that is, the effective key size gets doubled. It is, therefore, customary to take $K_1 = K_3$ and K_2 different from this common value. The actual key size also gets doubled with this choice—one doesn't have to remember K_3 separately. It is also a common practice to take h_1 and h_3 the encryption function (under $K_1 = K_3$) and h_2 the decryption function (under K_2). One often calls this particular triple encryption an E-D-E scheme.

A.2.4 Modes of Operation

In practice, the length of the message m to be encrypted need not equal the block length n of the block cipher f. One then has to break up m into blocks of some fixed length $n' \leqslant n$ and encrypt each block using the block cipher. In order to make the length of m an integral multiple of n', one may have to pad extra bits to m (say, zero bits at the end). It is often necessary to store the initial size of m in a separate block, say, after the last message block. In what follows, we shall assume that the input message m gives rise to l blocks $m_1, m_2, \ldots, m_l$ each of size n'. The corresponding ciphertext blocks $c_1, c_2, \ldots, c_l$ will also be of bit length n' each. The reason for choosing the block size $n' \leqslant n$ will be clear soon.

The ECB mode

The easiest way to encrypt multiple blocks $m_1, \ldots, m_l$ is to take $n' = n$ and encrypt each block m_i as $c_i := f_K(m_i)$. Decryption is analogous: $m_i := f_K^{-1}(c_i)$. This mode of operation of a block cipher is called the *electronic code-book* or the *ECB mode*. Algorithms A.7 and A.8 describe this mode.

Algorithm A.7 **ECB encryption**

Input: The plaintext blocks $m_1, \ldots, m_l$ and the key K.
Output: The ciphertext $c = c_1 \ldots c_l$.
Steps:
for $i = 1, \ldots, l$ { $c_i := f_K(m_i)$ }

Algorithm A.8 **ECB decryption**

Input: The ciphertext blocks $c_1, \ldots, c_l$ and the key K.
Output: The plaintext $m = m_1 \ldots m_l$.
Steps:
for $i = 1, \ldots, l$ { $m_i := f_K^{-1}(c_i)$ }

In this mode, identical message blocks encrypt to identical ciphertext blocks (under the same key), that is, partial information about the plaintext may be leaked out. The following three modes overcome this problem.

The CBC mode

In the *cipher-block chaining* or the *CBC mode*, one takes $n' = n$ and each plaintext block is first XOR-ed with the previous ciphertext block and then encrypted. In order to XOR the first plaintext block, one needs an n-bit *initialization vector* (*IV*). The IV need not be kept secret and may be sent along with the ciphertext blocks.

Algorithm A.9 **CBC encryption**

Input: The plaintext blocks $m_1, \ldots, m_l$, the key K and the IV.
Output: The ciphertext $c = c_1 \ldots c_l$.
Steps:
$c_0 := \text{IV}$.
for $i = 1, \ldots, l$ { $c_i := f_K(m_i \oplus c_{i-1})$. }

Algorithm A.10 **CBC decryption**

Input: The ciphertext blocks $c_1, \ldots, c_l$, the key K and the IV.
Output: The plaintext $m = m_1 \ldots m_l$.
Steps:
$c_0 := \text{IV}$.
for $i = 1, \ldots, l$ { $m_i := f_K^{-1}(c_i) \oplus c_{i-1}$. }

The CFB mode

In the *cipher feedback* or the *CFB mode*, one chooses $n' \in \{1, 2, \ldots, n\}$. In this mode, the plaintext blocks are not encrypted, but masked by XOR-ing with a stream of random keys generated from a (not necessarily secret) n-bit IV. In this sense, the CFB mode works like a stream cipher (see Section A.3).

Algorithm A.11 **CFB encryption**

Input: The plaintext blocks $m_1, \ldots, m_l$, the key K and the IV.
Output: The ciphertext $c = c_1 \ldots c_l$.
Steps:
$k_0 := \text{IV}$. /* Initialize the key stream */
for $i = 1, \ldots, l$ {
  /* Mask the current key by block encryption and the message by XOR-ing */
  $c_i := m_i \oplus \text{msb}_{n'}(f_K(k_{i-1}))$.
  /* Generate the next key from the previous key and the current ciphertext block */
  $k_i := \text{lsb}_{n-n'}(k_{i-1}) \,||\, c_i$.
}

Algorithm A.11 explains CFB encryption. The notation $\text{msb}_k(z)$ (resp. $\text{lsb}_k(z)$) stands for the most (resp. least) significant k bits of a bit string z. For CFB decryption

(Algorithm A.12), the identical key stream $k_0, k_1, \ldots, k_l$ is generated and used to mask off the message blocks from the ciphertext blocks.

Algorithm A.12 **CFB decryption**

Input: The ciphertext blocks $c_1, \ldots, c_l$, the key K and the IV.
Output: The plaintext $m = m_1 \ldots m_l$.
Steps:
$k_0 := \text{IV}$.
for $i = 1, \ldots, l$ {
  $m_i := c_i \oplus \text{msb}_{n'}(f_K(k_{i-1}))$.
  $k_i := \text{lsb}_{n-n'}(k_{i-1}) \,||\, c_i$.
}

The OFB mode

The *output feedback* or the *OFB mode* also works like a stream cipher by masking the plaintext blocks using a stream of keys. The key stream in the OFB mode is generated by successively applying the block encryption function on an n-bit (not necessarily secret) IV. Here, one chooses any $n' \in \{1, 2, \ldots, n\}$.

OFB encryption is explained in Algorithm A.13. OFB decryption (Algorithm A.14) is identical, with only the roles of m and c interchanged, and requires the generation of the same key stream $k_0, k_1, \ldots, k_l$ used during encryption.

Algorithm A.13 **OFB encryption**

Input: The plaintext blocks $m_1, \ldots, m_l$, the key K and the IV.
Output: The ciphertext $c = c_1 \ldots c_l$.
Steps:
$k_0 := \text{IV}$. /* Initialize the key stream */
for $i = 1, \ldots, l$ {
  $k_i := f_K(k_{i-1})$. /* Generate the next key in the stream */
  $c_i := m_i \oplus \text{msb}_{n'}(k_i)$. /* Mask the plaintext block */
}

Algorithm A.14 **OFB decryption**

Input: The ciphertext blocks $c_1, \ldots, c_l$, the key K and the IV.
Output: The plaintext $m = m_1 \ldots m_l$.
Steps:
$k_0 := \text{IV}$. /* Initialize the key stream */
for $i = 1, \ldots, l$ {
  $k_i := f_K(k_{i-1})$. /* Generate the next key in the stream */
  $m_i := c_i \oplus \text{msb}_{n'}(k_i)$. /* Remove the mask from the ciphertext block */
}

EXERCISE SET A.2

A.1 Let us use the notations of Algorithm A.2. For a message m and round keys K_i, we have the values V, L_i, R_i, W, c. For another message m' and another set of round keys K'_i, let us denote these values by V', L'_i, R'_i, W', c'. Show that if $m' = c$ and if $K'_i = K_{17-i}$ for $i = 1, \ldots, 16$, then $L'_i = R_{16-i}$ and $R'_i = L_{16-i}$ for all $i = 0, 1, \ldots, 16$. Deduce that in this case we have $c' = m$. (This shows that DES decryption is the same as DES encryption with the key schedule reversed.)

A.2 For a bit string z, let $\overline{z}$ denote the bit-wise complement of z. Deduce that $\mathrm{DES}_{\overline{K}}(\overline{m}) = \overline{\mathrm{DES}_K(m)}$, that is, complementing both the plaintext message and the key complements the ciphertext message. **[H]**

A.3 A DES key K is said to be *weak*, if the DES key schedule on K gives $K_1 = K_2 = \cdots = K_{16}$. Show that there are exactly four weak DES keys which in hexadecimal notation are:

```
0101 0101 0101 0101
FEFE FEFE FEFE FEFE
1F1F 1F1F 0E0E 0E0E
E0E0 E0E0 F1F1 F1F1
```

A.4 A DES key K is said to be *anti-palindromic*, if the DES key schedule on K gives $K_i = \overline{K_{17-i}}$ for all $i = 1, \ldots, 16$. Show that the following four DES keys (in hexadecimal notation) are anti-palindromic:

```
01FE 01FE 01FE 01FE
FE01 FE01 FE01 FE01
1FE0 1FE0 0EF1 0EF1
E01F E01F F10E F10E
```

A.5 Represent $\mathbb{F}_{2^8} = \mathbb{F}_2[X]/\langle f(X)\rangle$, where $f(X) = X^8 + X^4 + X^3 + X + 1$ (Section A.2.2).
(a) Show that multiplication by x (the octet 02) in $\mathbb{F}_{2^8}$ can be computed by a left shift followed conditionally (derive the condition) by XORing with the octet $1\mathrm{b}$.
(b) Design an algorithm for multiplying two elements of $\mathbb{F}_{2^8}$ using bit manipulations on octets only.

A.6 The multiplication of $\mathbb{F}_{2^8}$ can be made table-driven. Since this field contains 256 elements, a 256×256 array suffices to store all the products. That requires a storage of 64 kb. We can considerably reduce the storage by using discrete logs.
(a) Show that the multiplicative order of x (in $\mathbb{F}^*_{2^8}$) is 51.
(b) Show that $x + 1$ is a generator of $\mathbb{F}^*_{2^8}$.
(c) Write a computer program to generate the table of discrete logarithms of elements of $\mathbb{F}^*_{2^8}$ to the base $x + 1$ (Table A.6).
(d) Write a computer program to generate the table of powers of $x + 1$ (Table A.7).
(e) Design an algorithm for multiplying two elements of $\mathbb{F}_{2^8}$ using table lookup.

Table A.6 Discrete-log table for AES

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	–	00	19	01	32	02	1a	c6	4b	c7	1b	68	33	ee	df	03
1	64	04	e0	0e	34	8d	81	ef	4c	71	08	c8	f8	69	1c	c1
2	7d	c2	1d	b5	f9	b9	27	6a	4d	e4	a6	72	9a	c9	09	78
3	65	2f	8a	05	21	0f	e1	24	12	f0	82	45	35	93	da	8e
4	96	8f	db	bd	36	d0	ce	94	13	5c	d2	f1	40	46	83	38
5	66	dd	fd	30	bf	06	8b	62	b3	25	e2	98	22	88	91	10
6	7e	6e	48	c3	a3	b6	1e	42	3a	6b	28	54	fa	85	3d	ba
7	2b	79	0a	15	9b	9f	5e	ca	4e	d4	ac	e5	f3	73	a7	57
8	af	58	a8	50	f4	ea	d6	74	4f	ae	e9	d5	e7	e6	ad	e8
9	2c	d7	75	7a	eb	16	0b	f5	59	cb	5f	b0	9c	a9	51	a0
a	7f	0c	f6	6f	17	c4	49	ec	d8	43	1f	2d	a4	76	7b	b7
b	cc	bb	3e	5a	fb	60	b1	86	3b	52	a1	6c	aa	55	29	9d
c	97	b2	87	90	61	be	dc	fc	bc	95	cf	cd	37	3f	5b	d1
d	53	39	84	3c	41	a2	6d	47	14	2a	9e	5d	56	f2	d3	ab
e	44	11	92	d9	23	20	2e	89	b4	7c	b8	26	77	99	e3	a5
f	67	4a	ed	de	c5	31	fe	18	0d	63	8c	80	c0	f7	70	07

Table A.7 Power table for AES

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	01	03	05	0f	11	33	55	ff	1a	2e	72	96	a1	f8	13	35
1	5f	e1	38	48	d8	73	95	a4	f7	02	06	0a	1e	22	66	aa
2	e5	34	5c	e4	37	59	eb	26	6a	be	d9	70	90	ab	e6	31
3	53	f5	04	0c	14	3c	44	cc	4f	d1	68	b8	d3	6e	b2	cd
4	4c	d4	67	a9	e0	3b	4d	d7	62	a6	f1	08	18	28	78	88
5	83	9e	b9	d0	6b	bd	dc	7f	81	98	b3	ce	49	db	76	9a
6	b5	c4	57	f9	10	30	50	f0	0b	1d	27	69	bb	d6	61	a3
7	fe	19	2b	7d	87	92	ad	ec	2f	71	93	ae	e9	20	60	a0
8	fb	16	3a	4e	d2	6d	b7	c2	5d	e7	32	56	fa	15	3f	41
9	c3	5e	e2	3d	47	c9	40	c0	5b	ed	2c	74	9c	bf	da	75
a	9f	ba	d5	64	ac	ef	2a	7e	82	9d	bc	df	7a	8e	89	80
b	9b	b6	c1	58	e8	23	65	af	ea	25	6f	b1	c8	43	c5	54
c	fc	1f	21	63	a5	f4	07	09	1b	2d	77	99	b0	cb	46	ca
d	45	cf	4a	de	79	8b	86	91	a8	e3	3e	42	c6	51	f3	0e
e	12	36	5a	ee	29	7b	8d	8c	8f	8a	85	94	a7	f2	0d	17
f	39	4b	dd	7c	84	97	a2	fd	1c	24	6c	b4	c7	52	f6	01

A.7 Denote the multiplication of $A := \mathbb{F}_{2^8}[Y]/\langle Y^4 + 1\rangle$ by $\otimes$ (Section A.2.2).
(a) Let $\alpha = a_3y^3 + a_2y^2 + a_1y + a_0$ and $\beta = b_3y^3 + b_2y^2 + b_1y + b_0$ be elements of A and $\gamma = c_3y^3 + c_2y^2 + c_1y + c_0 = \alpha \otimes \beta$. Show that

$$\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{pmatrix} \odot \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix},$$

where the matrix arithmetic on the right side follows the arithmetic of $\mathbb{F}_{2^8}$.

(b) Verify that the inverse of the element of A represented by the word 03010102 (in hex) is $0b0d090e$.

A.8 **(a)** Show that Transform (A.3) can be represented as

$$\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \\ c_7 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix},$$

where the matrix arithmetic on the right side is that of $\mathbb{F}_2$.

(b) Let $\mathcal{M}$ denote the 8×8 matrix of Part (a). Prove that $\mathcal{M}$ is invertible over $\mathbb{F}_2$ with

$$\mathcal{M}^{-1} = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}.$$

(c) Conclude that the transformation $A \mapsto \mathrm{SubOctet}(A)$ is invertible.

A.9 **(a)** Argue that the transforms $\mathrm{SubState}$ and $\mathrm{ShiftRows}$ commute with one another.

(b) Show that $\mathrm{MixCols}^{-1}(\mathrm{AddKey}(S, L_0, L_1, L_2, L_3)) = \mathrm{AddKey}(\mathrm{MixCols}^{-1}(S), \mathrm{MixCols}^{-1}(L_0, L_1, L_2, L_3))$ for a suitable meaning of the application of $\mathrm{MixCols}^{-1}$ on four 32-bit keys L_0, L_1, L_2 and L_3.

(c) Conclude that one can obtain a decryption key schedule $K'_0, K'_1, \ldots, K'_{4N_r+3}$ in such a way that Algorithm A.15 correctly performs AES decryption. **[H]**

Algorithm A.15 **Equivalent form of AES decryption**

Input: The ciphertext message $C = \gamma_0\gamma_1 \ldots \gamma_{15}$ and the decryption key schedule $K'_0, K'_1, \ldots, K'_{4N_r+3}$.
Output: Plaintext message $M = \mu_0\mu_1 \ldots \mu_{15}$.
Steps:
Convert C to the state S. /* Use Transform (A.1) */
$S := \mathrm{AddKey}(S, K'_{4N_r}, K'_{4N_r+1}, K'_{4N_r+2}, K'_{4N_r+3})$.
for $i = N_r - 1, N_r - 2, \ldots, 0$ {
    $S := \mathrm{SubState}^{-1}(S)$.
    $S := \mathrm{ShiftRows}^{-1}(S)$.
    if $(i \neq 0)$ { $S := \mathrm{MixCols}^{-1}(S)$. }
    $S := \mathrm{AddKey}(S, K'_{4i}, K'_{4i+1}, K'_{4i+2}, K'_{4i+3})$.
}
Convert S to the message M. /* Use Transform (A.2) */

A.10 Show that a multiple encryption scheme with exactly k stages provides an effective security of $\lceil k/2 \rceil$ keys against the meet-in-the-middle attack.

A.11 Consider a message m broken into blocks $m_1, \ldots, m_l$, encrypted to $c_1, \ldots, c_l$ and sent to an entity.

(a) Suppose that during the transmission exactly one ciphertext block gets corrupted. Show that for the different modes of encryption, the numbers ν of blocks that are incorrectly decrypted due to this transmission error are as listed in the following table.

Mode	ν
ECB	1
CBC	$\leqslant 2$
CFB	$\leqslant 1 + \lceil n/n' \rceil$
OFB	1

(b) For each of the four modes, discuss the effects on decryption caused by the insertion or deletion of a ciphertext block during transmission (say, by an active adversary).

A.3 Stream Ciphers

A block cipher encrypts large blocks of data using a fixed key. A stream cipher, on the other hand, encrypts small blocks of data (typically bits or bytes) using different keys. The security of a stream cipher stems from the unpredictability of guessing the keys in the key stream. Here, we deal with stream ciphers that encrypt bit-by-bit.

Definition A.2 A stream cipher F encrypts a plaintext $m = m_1 m_2 \ldots m_l$ to a ciphertext $c = c_1 c_2 \ldots c_l$ using a key stream $k = k_1 k_2 \ldots k_l$, where each $m_i, c_i, k_i \in \mathbb{Z}_2$. F uses a function $f : \mathbb{Z}_2 \times \mathbb{Z}_2 \to \mathbb{Z}_2$ that yields $f(m_i, k_i) = c_i$. In order to effect unique decryption, the map $f_\kappa : \mathbb{Z}_2 \to \mathbb{Z}_2$, $\mu \mapsto f(\mu, \kappa)$, must be a bijection for each $\kappa \in \mathbb{Z}_2$. F encrypts and decrypts bit-by-bit using the formulas $c_i = f_{k_i}(m_i)$ and $m_i = f_{k_i}^{-1}(c_i)$. □

Example A.1 An obvious choice for f_κ is $f_\kappa(\mu) := \mu \oplus \kappa$, so that $f_\kappa^{-1}(\chi) = \chi \oplus \kappa$. Suppose that the bits $k_1, k_2, \ldots, k_l$ in the key stream are generated randomly and uniformly, independent of the plaintext bits. Let us assume that for an $i \in \{1, 2, \ldots, l\}$ the probability $\Pr(m_i = 0)$ is p, so that $\Pr(m_i = 1) = 1 - p$. Since $\Pr(k_i = 0) = \Pr(k_i = 1) = 1/2$, and m_i and k_i are independent, we have:

$$\begin{aligned} \Pr(c_i = 0) &= \Pr(m_i = 0, k_i = 0) + \Pr(m_i = 1, k_i = 1) \\ &= \Pr(m_i = 0)\Pr(k_i = 0) + \Pr(m_i = 1)\Pr(k_i = 1) \\ &= p \times (1/2) + (1 - p) \times (1/2) = 1/2. \end{aligned}$$

So $\Pr(c_i = 1)$ is $1/2$ too, that is, the two values of c_i are equally likely, irrespective of the probability p. This, in turn, implies that the ciphertext bit c_i provides absolutely no information about the plaintext bit m_i. In this sense, this stream cipher, called *Vernam's one-time pad*, offers *unconditional security*. □

Generating a truly random key stream of arbitrary length is a difficult problem. Moreover, the same key stream is used for decryption and has to be reproduced at the recipient's end. In view of these difficulties, Vernam's one-time pad is used only very rarely.

A practical solution is to use a *pseudorandom* key stream $k_1, k_2, k_3, \ldots$ generated from a secret key J of fixed *small* length. The bits in the pseudorandom stream should be sufficiently unpredictable and the length of J adequately large, so as to preclude the possibility of mounting a successful attack in feasible time.

Depending on how the key stream is generated from J, stream ciphers can be broadly classified in two categories. In a *synchronous stream cipher*, each key in the key stream is generated independent of any plaintext or ciphertext bit, whereas in a *self-synchronizing* (or *asynchronous*) *stream cipher* each key in the stream is generated based only on J and a fixed number of previous ciphertext bits. Algorithms A.16 and A.17 explain the workings of these two classes of stream ciphers.

Algorithm A.16 **Encryption in a synchronous stream cipher**

Input: The message $m = m_1 m_2 \ldots m_l$, the secret key J and a (not necessarily secret) initial state S of the key stream generator.
Output: The ciphertext $c = c_1 c_2 \ldots c_l$.
Steps:

$s_0 := S$. /* Initialize the state of the key stream generator */
for $i = 1, \ldots, l$ {
  $k_i := g(s_{i-1}, J)$. /* Generate the key k_i */
  $s_i := \delta(s_{i-1}, J)$. /* Transition to the next state */
  $c_i := f_{k_i}(m_i)$. /* Encrypt the plaintext bit m_i */
}

Algorithm A.17 **Encryption in an asynchronous stream cipher**

Input: The message $m = m_1 m_2 \ldots m_l$, the secret key J and a (not necessarily secret) initial state $(c_{-t+1}, c_{-t+2}, \ldots, c_0)$.
Output: The ciphertext $c = c_1 c_2 \ldots c_l$.
Steps:

for $i = 1, \ldots, l$ {
  $k_i := g(c_{i-t}, c_{i-t+1}, \ldots, c_{i-1}, J)$. /* Generate the key k_i */
  $c_i := f_{k_i}(m_i)$. /* Encrypt the plaintext bit m_i */
}

A block cipher in the OFB mode works like a synchronous stream cipher, whereas a block cipher in the CFB mode like an asynchronous stream cipher.

A.3.1 Linear Feedback Shift Registers

Linear feedback shift registers (*LFSRs*), being suitable for hardware implementation and possessing good cryptographic properties, are widely used as basic building blocks

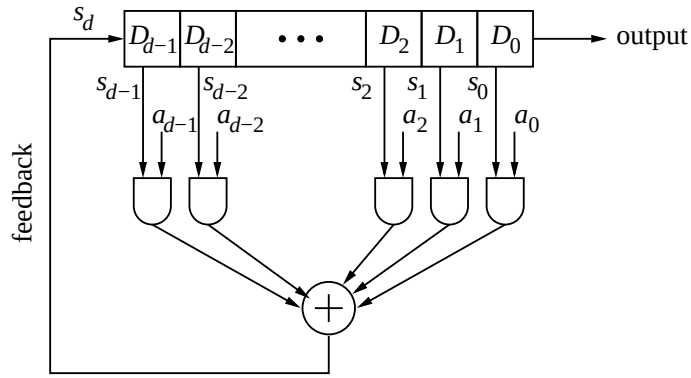


Figure A.2 A linear feedback shift register (LFSR) with d stages

for many stream ciphers. Figure A.2 depicts an LFSR L with d *stages* or *delay elements* $D_0, D_1, \ldots, D_{d-1}$, each capable of storing one bit. The state of the LFSR is described by the d-tuple $\mathbf{s} := (s_0, s_1, \ldots, s_{d-1})$, where s_i is the bit stored in D_i. It is often convenient to treat **s** as the column vector $(s_0\ s_1\ \cdots\ s_{d-1})^{\mathrm{t}}$.

There are d control bits $a_0, a_1, \ldots, a_{d-1}$. The working of the LFSR is governed by a clock. At every clock pulse the bits stored in the delay elements are bit-wise AND-ed with the respective control bits and the AND gate outputs are XOR-ed to obtain the bit s_d. The bit s_0 stored in D_0 is delivered to the output. Finally, for each $i \in \{0, 1, \ldots, d-1\}$ the delay element D_i sets its stored bit to s_{i+1}, that is, the register experiences a right shift by one bit with the feedback bit s_d filling up the leftmost delay element.

Thus, a clock pulse changes the state of the LFSR from $\mathbf{s} := (s_0, s_1, \ldots, s_{d-1})$ to $\mathbf{t} := (t_0, t_1, \ldots, t_{d-1})$, where **s** and **t** are related as:

$$\begin{aligned} t_0 &= s_1 \\ t_1 &= s_2 \\ &\vdots \\ t_{d-2} &= s_{d-1} \\ t_{d-1} &\equiv a_0 s_0 + a_1 s_1 + a_2 s_2 + \cdots + a_{d-1} s_{d-1} \pmod 2. \end{aligned}$$

If **s** and **t** are treated as column vectors, this can be compactly represented as

$$\mathbf{t} \equiv \Delta_L \mathbf{s} \pmod 2, \tag{A.4}$$

where the transition matrix Δ_L is given by

$$\Delta_L := \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 1 \\ a_0 & a_1 & a_2 & \cdots & a_{d-2} & a_{d-1} \end{pmatrix}. \tag{A.5}$$

When the LFSR L is initialized to a non-zero state, the bit stream output by it can be used as a pseudorandom bit sequence. For a given set of control bits $a_0, \ldots, a_{d-1}$,

the next state of L is uniquely determined by its previous state only. Since L has only finitely many $(2^d - 1)$ non-zero states, the output bit sequence of L must be (eventually) periodic. For cryptographic use, the period of the bit sequence should be as large as possible. If the period is maximum possible, namely $2^d - 1$, L is called a *maximum-length LFSR*.

Many properties of the LFSR L can be explained in terms of its *connection polynomial* defined as:

$$C_L(X) := 1 + a_{d-1}X + a_{d-2}X^2 + \cdots + a_1X^{d-1} + a_0X^d \in \mathbb{F}_2[X]. \qquad \textbf{(A.6)}$$

For example, assume that $a_0 = 1$, so that $\deg C_L(X) = d$. Assume further that $C_L(X)$ is irreducible (over $\mathbb{F}_2$). Consider the extension $\mathbb{F}_{2^d}$ of $\mathbb{F}_2$, represented as $\mathbb{F}_2[X]/\langle C_L(X)\rangle = \mathbb{F}_2(x)$, where $x := X + \langle C_L(X)\rangle \in \mathbb{F}_{2^d}$. It turns out that if x is a generator of the cyclic group $\mathbb{F}^*_{2^d}$, then L is a maximum-length LFSR. In this case, the polynomial $C_L(X)$ is called a *primitive polynomial* of $\mathbb{F}_2[X]$.[3]

A.3.2 Stream Ciphers Based on LFSRs

The bit sequence output by an LFSR L can be used as the key stream $k_1k_2\ldots k_l$ in order encrypt a plaintext stream $m_1m_2\ldots m_l$ to the ciphertext stream $c_1c_2\ldots c_l$ with $c_i := m_i \oplus k_i$. The number d of stages in L should be chosen reasonably large and the control bits $a_0, \ldots, a_{d-1}$ should be kept secret. The initial state of L may or may not be a secret. For suitable choices of $a_0, \ldots, a_{d-1}$, the output sequences from L possess good statistical properties and hence L appears to be an efficient key stream generator.

Unfortunately, such a key stream generator is vulnerable to a known-plaintext attack as follows. Suppose that m_i and c_i are known for $i = 1, 2, \ldots, 2d$. One can easily compute $k_i = m_i \oplus c_i$ for all these i. Let $\mathbf{s}_i := (k_i, k_{i+1}, \ldots, k_{i+d-1})$ denote the state of L while outputting c_i. By Congruence (A.4), $\mathbf{s}_{i+1} \equiv \Delta_L\mathbf{s}_i \pmod 2$ for $i = 1, 2, \ldots, d$. Define the $d \times d$ matrices $S := (\mathbf{s}_1 \ \ \mathbf{s}_2 \ \ \cdots \ \ \mathbf{s}_d)$ and $T := (\mathbf{s}_2 \ \ \mathbf{s}_3 \ \ \cdots \ \ \mathbf{s}_{d+1})$, where $\mathbf{s}_i$ are treated as column vectors as before. We then have $T \equiv \Delta_L S \pmod 2$. If S is invertible modulo 2, then Δ_L and hence the secret control bits can be easily computed. In order to avoid this known-plaintext attack, one should introduce some non-linearity in the LFSR outputs.

A *non-linear combination generator* combines the output bits $u_1, u_2, \ldots, u_r$ from r LFSRs by a non-linear function φ in order to generate the key $k = \varphi(u_1, \ldots, u_r)$. The *Geffe generator* of Figure A.3 gives a well-known example. It uses the non-linear function $\varphi(u_1, u_2, u_3) = (u_1u_2)\oplus(\overline{u_2}u_3)$, that is, $\varphi(u_1, u_2, u_3) \equiv u_1u_2+(1+u_2)u_3 \equiv u_1u_2 + u_2u_3 + u_3 \pmod 2$.

A *non-linear filter generator* generates the key as $k = \psi(s_0, s_1, \ldots, s_{d-1})$, where $s_0, \ldots, s_{d-1}$ are the bits stored in the delay elements of a single LFSR and where ψ is a non-linear function.

Several other ad hoc schemes can destroy the linearity of an LFSR's output. The *shrinking generator*, for example, uses two LFSRs L_1 and L_2. Both L_1 and L_2 are simultaneously clocked. If the output of L_1 is 1, the output of L_2 goes to the key stream, whereas if the output of L_1 is 0, the output of L_2 is discarded. The resulting

[3] A primitive polynomial defined in this way has nothing to do with a primitive polynomial over a UFD, defined in Exercise 2.54. Mathematicians often go for such multiple definitions of the same terms and phrases.

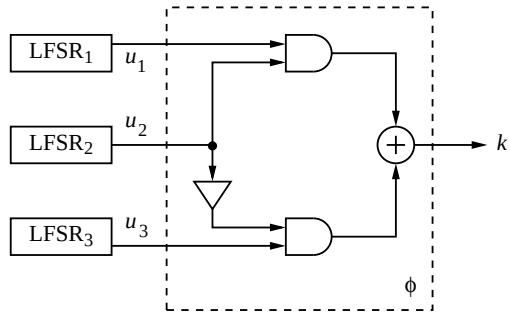


Figure A.3 The Geffe generator

key stream is an irregularly (and non-linearly) decimated subsequence of the output sequence of L_2.

The non-linear function (φ or ψ) eliminates the chance of mounting the straightforward known-plaintext attack described above. However, for polynomial non-linearities certain algebraic attacks are known, for example, see Courtois and Pieprzyk [67, 66].[4] Solving non-linear polynomial equations is usually more difficult than solving linear equations, but ample care should be taken to avoid accidental encounters with easily solvable systems. Complacency is a word ever excluded from a cryptologer's world.

EXERCISE SET A.3

A.12 For each of the two classes of stream ciphers (Algorithms A.16, A.17) discuss the effects on decryption of
(a) alteration
(b) insertion or deletion
of a ciphertext bit during transmission.

A.13 Suppose that the LFSR L of Figure A.4 is initialized to the state $(1, 0, 0, 0)$. Derive the sequence of state transitions of the LFSR, and hence determine the output bit sequence of L. Argue that L is a maximum-length LFSR. Verify (according to the definition) that the connection polynomial $C_L(X)$ is primitive.

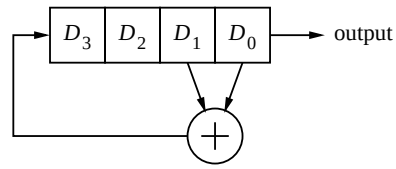


Figure A.4 An LFSR with four stages

[4] Visit the Internet site http://www.cryptosystem.net/ for more papers in related areas.

A.14 Let Δ_L and $C_L(X)$ be as in Equations (A.5) and (A.6). Show that:
(a) Δ_L is invertible modulo 2 if and only if $a_0 = 1$.
(b) The characteristic polynomial of Δ_L (a matrix over $\mathbb{F}_2$) is $X^d C_L(1/X)$. **[H]**

A.15 Let L be an LFSR with connection polynomial $C_L(X)$. Further let $S(X) := s_0 + s_1 X + s_2 X^2 + \cdots = \sum_{i \in \mathbb{N}_0} s_i X^i$, $s_i \in \mathbb{F}_2$, denote a power series[5] over $\mathbb{F}_2$. Show that L generates the (infinite) bit sequence $s_0, s_1, s_2, \ldots$ if and only if the product $C_L(X)S(X)$ modulo 2 is a polynomial of degree $< d$.

A.16 Let $\sigma = s_0 s_1 \ldots s_{d-1} \neq 00\ldots0$ be a bit string of length $d \geqslant 1$. The *linear complexity* $L(\sigma)$ of σ is defined to be the length of the shortest LFSR that generates σ as the leftmost part of its output (after it is initialized to a suitable state). Prove that:
(a) $L(\sigma) \leqslant d$.
(b) $L(\sigma) = d$ if and only if $\sigma = 00\ldots01$. **[H]**

A.17 Assume that the three LFSR outputs u_1, u_2, u_3 in the Geffe generator are uniformly distributed. Show that $\Pr(k = u_1) = 3/4 = \Pr(k = u_3)$. Thus, partial information about the internal details of the Geffe generator is leaked out in the key stream.

A.4 Hash Functions

A *hash function* maps bit strings of any length to bit strings of a fixed length n. For practical uses, hash functions should be easy to compute, that is, computing the hash of x should be doable in time polynomial in the size of x.

Since a hash function H maps an infinite set to a finite set, there *must* exist pairs (x_1, x_2) of *distinct* strings with $H(x_1) = H(x_2)$. Such a pair is called a *collision* for H. For cryptographic applications (for example, for generating digital signatures), it should be computationally infeasible to find collisions for hash functions. To elaborate this topic further we mention the following two desirable properties of hash functions used in cryptography.

Definition A.3 A hash function H is called *second pre-image resistant*, if it is computationally infeasible[6] to find, for a *given* bit string x_1, a second bit string x_2 with $H(x_1) = H(x_2)$. □

Definition A.4 A hash function H is called *collision resistant*, if it is computationally infeasible to find any two distinct bit strings x_1 and x_2 with $H(x_1) = H(x_2)$. □

[5] A *power series* over a ring A is a (formal) expression of the form $a_0 + a_1 X + a_2 X^2 + \cdots = \sum_{i \in \mathbb{N}_0} a_i X^i$ with each $a_i \in A$. The set of all such power series is denoted by $A[[X]]$. For two power series $f := \sum_{i \in \mathbb{N}_0} a_i X^i$ and $g := \sum_{i \in \mathbb{N}_0} b_i X^i$ over A, the sum $f + g$ is defined to be the power series $\sum_{i \in \mathbb{N}_0} (a_i + b_i) X^i$ and the product fg is defined as the power series $\sum_{i \in \mathbb{N}_0} c_i X^i$, where $c_i := \sum_{k=0}^{i} a_k b_{i-k}$. Under these operations $A[[X]]$ is a ring. A polynomial over A can be identified with an element of $A[[X]]$, in which all, but finitely many, coefficients are zero.

[6] A problem P is said to be computationally infeasible if any known or possible algorithm (deterministic or randomized) to solve P runs in infeasible (like super-polynomial) time, except perhaps for a set of some input instances, the density of which in the input space is zero (or, more generally, negligibly small).

In order to prevent existential forgery (Exercise 5.15) of digital signatures, hash functions should also be difficult to invert.

Definition A.5 An n-bit hash function H is called *first pre-image resistant* (or simply *pre-image resistant*), if it is computationally infeasible to find, for almost all bit strings y of length n, a bit string x (of any length) such that $y = H(x)$. The qualification *almost all* in the last sentence was necessary, since one can compute and store the pairs $(x_i, H(x_i))$, $i = 1, 2, \ldots, k$, for some *small* k and for some x_i of one's choice. If the given y turns out to be one of these hash values $H(x_i)$, a pre-image of y is easily available. □

A hash function (provably or believably) satisfying all these three properties is called a *cryptographic hash function*. A hash function having first and second pre-image resistance is often called a *one-way hash function*. Some authors require both second pre-image resistance and collision resistance to define a *collision-resistant hash function*, but here we stick to Definitions A.3 and A.4. In what follows, an unqualified use of the phrase *hash function* indicates a *cryptographic hash function*.

Most of the properties of a cryptographic hash function are mutually independent. However, we have the following implication.

Proposition A.1 A collision resistant hash function is second pre-image resistant.

Proof Let H be a (non-cryptographic) hash function which is not second pre-image resistant. This means that there is an algorithm A that efficiently computes second pre-images, except perhaps for a vanishingly small fraction of inputs. Choose a random bit string x_1. The probability that x_1 is not a bad input to A is very high and, in that case, A outputs a second pre-image x_2 quickly. This gives us an efficient randomized algorithm to compute collisions (x_1, x_2) for H. ■

The converse of Proposition A.1 is not true: A second pre-image resistant hash function need *not* be collision resistant (Exercise A.19). Also collision resistance (or second pre-image resistance) does not imply first pre-image resistance (Exercise A.20), and first pre-image resistance does not imply second pre-image resistance (Exercise A.21).

A hash function may or may not be used in conjunction with a secret key. An *unkeyed hash function* is typically used to check the integrity of a message and is often called a *modification detection code* (*MDC*). A *keyed hash function*, on the other hand, is usually employed to authenticate the origin of a message (in addition to verifying the integrity of the message) and so is often called a *message authentication code* (*MAC*).

A.4.1 Merkle's Meta Method

Let us now describe a generic method of constructing hash functions. We start by defining the following basic building block.

Definition A.6 Let $m, n \in \mathbb{N}$ with $m = n + r$ for some $r \in \mathbb{N}$. A function $F : \mathbb{Z}_2^m \to \mathbb{Z}_2^n$ that maps bit strings of length m to bit strings of length n is called a *compression function*. Henceforth, we will consider only those compression functions that can be computed easily, that is, in polynomial time of the input size. □

Since $m > n$, collisions must exist for F. For cryptographic use, collisions should be difficult to locate. We can define first and second pre-image resistance and collision resistance of compression functions as before.

Algorithm A.18 **Merkle's meta method**

Input: A compression function $F : \mathbb{Z}_2^m \to \mathbb{Z}_2^n$ with $m = n + r$ and a bit string x of length $< 2^r$.
Output: The hash value $H(x)$.
Steps:
Let λ be the bit length of x.
Set $l := \lceil \lambda / r \rceil$.
If (λ is not a multiple of r) { Append $rl - \lambda$ zero bits to the right of x. }
Break the padded x into blocks $x_1, \ldots, x_l$ each of length r.
Store in a new block x_{l+1} the r-bit representation of λ.
Initialize $h_0 := 0^r$.
for $i = 1, 2, \ldots, l + 1$ { $h_i := F(h_{i-1} \,||\, x_i)$ }
Set $H(x) := h_{l+1}$.

Algorithm A.18 demonstrates how a compression function can be used to design an n-bit hash function H. The input message x is first broken into $l \geqslant 0$ blocks each of bit length r, after padding zero bits, if necessary. The initial bit length λ of x is then stored in a new block. This implies that H cannot handle bit strings of length $\geqslant 2^r$. For a reasonably big r, this is not a practical limitation. Storing λ is necessary for several reasons. First, it ensures that the for loop is executed at least once for any message. This prevents the trivial hash value 0^r (the bit string of length r containing zero bits only) for the null message. Moreover, if $h_i = 0^r$ for some $i \in \{1, \ldots, l\}$, then, without the length block, we would get $H(x_1 \,||\, \cdots \,||\, x_l) = H(x_{i+1} \,||\, \cdots \,||\, x_l)$ that leads to a collision for H.

We now show if F possesses the desired properties for use in cryptography, then so does H too.

Proposition A.2 If F is first pre-image resistant, then so is H.

Proof Assume that H is not first pre-image resistant, that is, an efficient algorithm A exists to compute x with $H(x) = y$ for most (if not all) $y \in \mathbb{Z}_2^n$. Since $y = h_{l+1} = F(h_l \,||\, x_{l+1})$, a pre-image (namely, $h_l \,||\, x_{l+1}$) of y under F is easily computable. ■

Proposition A.3 If F is collision resistant, then H is collision resistant (and hence also second pre-image resistant).

Proof Given a collision (x, x') for H, we can find a collision for F with little additional effort. We use the notations of Algorithm A.18 with primed variables for x'.

First consider $l \neq l'$. But then, in particular, the length blocks x_{l+1} and $x'_{l'+1}$ are different and thus $(h_l \,||\, x_{l+1}, h'_{l'} \,||\, x'_{l'+1})$ is a collision for F. So for the rest of the proof we take $l = l'$.

Now, suppose that $h_i \neq h'_i$ for some $i \in \{1, 2, \ldots, l\}$. Choose the largest such i and note that h_{i+1} and h'_{i+1} are defined and equal for this choice. This gives us the collision $(h_i \,||\, x_{i+1}, h'_i \,||\, x'_{i+1})$ for F.

The only case that remains to be treated is $h_i = h'_i$ for all $i \in \{0, 1, \ldots, l + 1\}$. Since $x \neq x'$, there is at least one $i \in \{1, 2, \ldots, l\}$ with $x_i \neq x'_i$. For such an i, the equality $h_i = h'_i$ implies that $(h_{i-1} \,||\, x_i, h'_{i-1} \,||\, x'_i)$ is a collision for F. ■

In order to design cryptographic hash functions, it suffices to design cryptographic compression functions. Block ciphers can be used for that purpose. Let f be a block cipher with block size n and key size r. Take $m := n + r$ and consider the map $F : \mathbb{Z}_2^m \to \mathbb{Z}_2^n$ that sends $x = L \,||\, R$ with $L \in \mathbb{Z}_2^n$ and $R \in \mathbb{Z}_2^r$ to the encrypted bit string $f_R(L)$. If f_R are assumed to be random permutations of $\mathbb{Z}_2^n$, the resulting compression function F possesses the desirable properties.

A.4.2 The Secure Hash Algorithm

Several custom-designed hash functions have been popularly used by the cryptography community. MD4 and MD5 are somewhat older 128-bit hash functions. Soon after its conception, MD4 was found to be vulnerable to several attacks. Also collisions for the compression function of MD5 are known. Therefore, these two hash functions have lost the desired level of confidence for cryptographic uses.

NIST has proposed a family of four hash algorithms. These algorithms are called *secure hash algorithms* and have the short names SHA-1, SHA-256, SHA-384 and SHA-512, which respectively produce 160-, 256-, 384- and 512-bit hash values. No collisions for SHA are known till date. In the rest of this section, we explain the SHA-1 algorithm. The workings of the other SHA algorithms are very similar and can be found in the FIPS document [222]. RIPEMD-160 is another popular 160-bit hash function.

SHA-1 (like other custom-designed hash functions mentioned above) is suitable for implementation in 32-bit processors. Suppose that we want to compute the hash $\text{SHA-1}(M)$ of a message M of bit length λ. First, M is padded to get the bit string $M' := M \,||\, 1 \,||\, 0^k \,||\, \Lambda$, where Λ is the 64-bit representation of λ, and where k is the smallest non-negative integer for which the bit length of M', that is, $\lambda + 1 + k + 64$, is a multiple of 512. M' is broken into blocks $M^{(1)}, M^{(2)}, \ldots, M^{(l)}$ each of length 512 bits. Each $M^{(i)}$ is represented as a collection of sixteen 32-bit words $M_j^{(i)}$, $j = 0, 1, \ldots, 15$. SHA-1 supports big-endian packing, that is, $M_0^{(i)}$ stores the leftmost 32 bits of $M^{(i)}$, $M_1^{(i)}$ the next 32 bits of $M^{(i)}$, $\ldots$, $M_{15}^{(i)}$ the rightmost 32 bits of $M^{(i)}$.

The SHA-1 computations are given in Algorithm A.19. One starts with a fixed initial 160-bit hash $H^{(0)}$. Successively for $i = 1, 2, \ldots, l$ the i-th message block $M^{(i)}$ is considered and the previous hash value $H^{(i-1)}$ is updated to $H^{(i)}$. At the end of the loop the 160-bit string $H^{(l)}$ is returned as $\text{SHA-1}(M)$. Each $H^{(i)}$ is represented by five 32-bit words $H_j^{(i)}$, $j = 0, 1, 2, 3, 4$. Here also, big-endian notation is used, that is, $H_0^{(i)}$ stores the leftmost 32 bits of $H^{(i)}$, $\ldots$, $H_4^{(i)}$ the rightmost 32 bits of $H^{(i)}$.

The updating procedure uses logical functions f_j. Here, product (like xy) implies bit-wise AND, bar (as in $\overline{x}$) denotes bit-wise complementation and $\oplus$ denotes bit-wise XOR, each on 32-bit operands. The notation $\text{LR}^k(z)$ (resp. $\text{RR}^k(z)$) stands for a left (resp. right) rotation, that is, a cyclic left (resp. right) shift, of the bit string z of length 32 by k positions.

The bits of $H^{(i)}$ are well-defined transformations of the bits of $H^{(i-1)}$ under the guidance of the bits of $M^{(i)}$. The good amount of non-linearity, introduced by the functions f_j and the modulo 2^{32} sums, makes it difficult to invert the transformation $H^{(i-1)} \mapsto H^{(i)}$ and thereby makes SHA-1 an (apparently) secure hash function.

Algorithm A.19 **The SHA-1 algorithm**

Input: A message M.
Output: The hash SHA-1(M) of M.
Steps:
Generate the message blocks $M^{(i)}$, $i = 1, 2, \ldots, l$.
/* Initialize the hash value */
$H_0 :=$ `0x67452301 efcdab89 98badcfe 10325476 c3d2e1f0`.
for $i = 1, 2, \ldots, l$ {
  /* Compute the message schedule W_j, $0 \leqslant j \leqslant 79$. */
  for $j = 0, 1, \ldots, 15$ { $W_j := M_j^{(i)}$ }
  for $j = 16, 17, \ldots, 79$ { $W_j := \mathrm{LR}^1(W_{j-3} \oplus W_{j-8} \oplus W_{j-14} \oplus W_{j-16})$ }
  /* Store the previous hash words */
  for $j = 0, 1, 2, 3, 4$ { $t_j := H_j^{(i-1)}$ }
  /* Compute the updating values */
  for $j = 0, 1, \ldots, 79$ {

$$T := \left(\mathrm{LR}^5(t_0) + f_j(t_1, t_2, t_3) + t_4 + K_j + W_j\right) \text{ rem } 2^{32}, \text{ where}$$

$$f_j(x, y, z) = \begin{cases} xy \oplus \overline{x}z & \text{if } 0 \leqslant j \leqslant 19 \\ x \oplus y \oplus z & \text{if } 20 \leqslant j \leqslant 39 \\ xy \oplus xz \oplus yz & \text{if } 40 \leqslant j \leqslant 59 \\ x \oplus y \oplus z & \text{if } 60 \leqslant j \leqslant 79 \end{cases}$$

and

$$K_j = \begin{cases} \texttt{0x5a827999} & \text{if } 0 \leqslant j \leqslant 19 \\ \texttt{0x6ed9eba1} & \text{if } 20 \leqslant j \leqslant 39 \\ \texttt{0x8f1bbcdc} & \text{if } 40 \leqslant j \leqslant 59 \\ \texttt{0xca62c1d6} & \text{if } 60 \leqslant j \leqslant 79 \end{cases}$$

$t_4 := t_3,\ t_3 := t_2,\ t_2 := \mathrm{RR}^2(t_1),\ t_1 := t_0,\ t_0 := T.$
  }
  /* Update the hash value */
  for $j = 0, 1, 2, 3, 4$ { $H_j^{(i)} := \left(t_j + H_j^{(i-1)}\right) \text{ rem } 2^{32}$ }
}
Set SHA-1$(M) := H^{(l)}$.

A test vector for SHA-1 is the following (here `616263` is the string "abc"):

SHA-1(`616263`) = `a9993e364706816aba3e25717850c26c9cd0d89d`.

EXERCISE SET A.4

A.18 Let x be a bit string. Break up x into blocks $x_1, \ldots, x_l$ each of bit size n (after padding, if necessary). Define $H_1(x) := x_1 \oplus \cdots \oplus x_l$. Show that H_1 possesses none of the desirable properties of a cryptographic hash function.

A.19 Let H be an n-bit cryptographic hash function and S a *finite* set of strings with $\#S \geqslant 2$. Define the function $H_2(x) := \begin{cases} 0^{n+1} & \text{if } x \in S, \\ 1 \,||\, H(x) & \text{otherwise.} \end{cases}$ Here, 0^{n+1} refers to a bit string of length $n+1$ containing zero-bits only. Show that H_2 is second pre-image resistant, but not collision resistant. **[H]**

A.20 Let H be an n-bit cryptographic hash function. Show that the function H_3 defined as $H_3(x) := \begin{cases} 0 \,||\, x & \text{if the bit length of } x \text{ is } n, \\ 1 \,||\, H(x) & \text{otherwise,} \end{cases}$ is collision resistant (and hence second pre-image resistant), but not first pre-image resistant. **[H]**

A.21 Let m be a product of two (unknown) big primes and let the binary representation of m (with leading one-bit) have n bits. Assume that it is computationally infeasible to compute square roots modulo m. We can identify bit strings with integers in a natural way. For a bit string x, take $y := 1 \,||\, x$ and let $H_4(x)$ denote the n-bit binary representation of $y^2 \pmod m$. Show that H_4 is first pre-image resistant, but not second pre-image resistant (and hence not collision-resistant). **[H]**

A.22 Let H be an n-bit cryptographic hash function. Assume that H produces random hash values on random input strings. Prove that $O(2^{n/2})$ hash values need to be computed to detect a collision for H with high probability. **[H]** Deduce also that nearly 2^{n-1} hash values need to be computed on an average to obtain a second pre-image x' of $H(x)$.

A.23 Let $F_1 : \mathbb{Z}_2^{2n} \to \mathbb{Z}_2^n$ be a *collision resistant* compression function.
(a) Define a compression function $F_2 : \mathbb{Z}_2^{4n} \to \mathbb{Z}_2^n$ as follows. Let x be a bit string of length $4n$. Write $x = L \,||\, R$, where each of L and R is of length $2n$ bits. Define $F_2(x) := F_1(F_1(L) \,||\, F_1(R))$. Show that F_2 is also collision-resistant.
(b) Inductively define $F_k : \mathbb{Z}_2^{2^k n} \to \mathbb{Z}_2^n$ as $F_k(x) := F_1(F_{k-1}(L) \,||\, F_{k-1}(R))$, where L and R are the left and right halves of x. Show that each F_k is collision resistant.
(c) Show that if F_1 is first pre-image resistant, then so is each F_k.
(d) Define an n-bit hash function H as follows. Let x be a bit string of length l. If $l < n$, take $k := 1$, else choose $k \in \mathbb{N}$ such that $2^{k-1}n \leqslant l < 2^k n$. Construct the string $y := x \,||\, 1 \,||\, 0^{2^k n - l - 1} \in \mathbb{Z}_2^{2^k n}$ and define $H(x) := F_k(y)$. Is H collision resistant? **[H]** (Appending a one-bit at the end of x delimits x and thereby prevents trivial collisions.)

A.24 **(a)** Let $F_1 : \mathbb{Z}_2^{m_1} \to \mathbb{Z}_2^{n_1}$ and $F_2 : \mathbb{Z}_2^{m_2} \to \mathbb{Z}_2^{n_2}$ be cryptographic compression functions. Show that $F : \mathbb{Z}_2^{m_1+m_2} \to \mathbb{Z}_2^{n_1+n_2}$ defined as $F(L \,||\, R) := F_1(L) \,||\, F_2(R)$ (where $L \in \mathbb{Z}_2^{m_1}$ and $R \in \mathbb{Z}_2^{m_2}$) is again a cryptographic compression function.
(b) The hash function H derived from DES (Section A.4.1) produces 64-bit hash values. For reasonable security, we require n-bit hash values with n at least 128. Use Part (a) to propose a method to make H achieve this desired level of security.

A.25 Assume that in the SHA-1 algorithm the designers opted for Algorithm A.19 with the following *minor* modifications: They defined f_j as $f_j(x, y, z) := x \oplus y \oplus z$ for all $j \in \{0, 1, \ldots, 79\}$ and they replaced all *costly* mod 2^{32} addition operations ($+$) by *cheap* bit-wise XOR operations ($\oplus$). Do you sense anything wrong with this design? **[H]**

Key Exchange in Sensor Networks

B

B.1 Introduction
B.2 Security Issues in a Sensor Network
B.3 The Basic Bootstrapping Framework
B.4 The Basic Random Key Predistribution Scheme
B.5 Random Pairwise Scheme
B.6 Polynomial-pool-based Key Predistribution
B.7 Matrix-based Key Predistribution
B.8 Location-aware Key Predistribution

One of the keys to happiness is a bad memory.
— Rita Mae Brown

That theory is worthless. It isn't even wrong!
— Wolfgang Pauli

You're only as sick as your secrets.
— Anonymous

B.1 Introduction

Public-key cryptography is not a solution to every security problem. Asymmetric routines are bulky and slow, and, in practice, augment symmetric cryptography by eliminating the need for prior secret establishment of keys between communicating parties. On a workstation of today's computing technology, this is an interesting and acceptable breakthrough. A 1 GHz processor runs one public-key encryption or key-exchange primitive in tens to hundreds of milliseconds, using at least hundreds of kilobytes of memory. That is reasonable for most applications, given that the routines are invoked rather infrequently.

Now, imagine a situation, where many tiny computing nodes, called *sensor nodes*, are scattered in an area for the purpose of sensing some data and transmitting the data to nearby *base stations* for further processing. This transmission is done by short-range radio communications. The base stations are assumed to be computationally well-equipped, but the sensor nodes are resource-starved. Such networks of sensor nodes are used in many important applications including tracking of objects in an enemy's area for military purposes and scientific, engineering and medical explorations like wildlife monitoring, distributed seismic measurement, pollution tracking, monitoring fire and nuclear power plants and tracking patients. In some cases, mostly for military and medical applications, data collected by sensor nodes need to be encrypted before transmitting to neighbouring nodes and base stations.

Evidently one has to resort to symmetric-key cryptography in order to meet the security needs in a sensor network. Appendix B provides an overview of some key exchange schemes suitable for sensor networks.

B.2 Security Issues in a Sensor Network

Several issues make secure communication in sensor networks different from that in usual networks:

Limited resources in sensor nodes Each sensor node contains a primitive processor featuring very low computing speed and only small amount of programmable memory. The popular Atmel ATmega 128L processor, as an example, has an 8-bit 4 MHz RISC processor and only 128 kbytes of RAM. The processor does not support instructions for multiplying or dividing integers. One requires tens of minutes to several hours for performing a single RSA or Diffie–Hellman exponentiation for cryptographic key sizes.

Limited lifetime of sensor nodes Each sensor node is battery-powered and is expected to operate for only a few days. Once deployed sensor nodes die, it becomes necessary to add fresh nodes to the network for continuing the data collection operation. This calls for dynamic management of security objects (like keys).

Limited communication ability of sensor nodes Sensor nodes communicate with each other and the base stations by wireless radio transmission at low bandwidth and over small communication ranges. For the Atmel ATmega 128L processor, the maximum bandwidth is 40 kbps, and the communication range is at most 100 feet (30 m).

Moreover, the deployment area may have irregularities (like physical obstacles) that further limit the communication abilities of the nodes. One, therefore, expects that a deployed sensor node can directly communicate with only few other nodes in the network.

Possibility of node capture A sensor network is vulnerable to capture of nodes by the enemy. The captured nodes may be physically destroyed or utilized to send misleading signals and/or disrupt the normal activity of the network. As a result, no node should have full trust on the nodes with which it communicates. The relevant security goal in this context is that the captured nodes should not divulge to the enemy enough secrets to jeopardize the communication among the uncaptured nodes.

Lack of knowledge about deployment configuration In many situations (like scattering of nodes from airplanes or trucks), the post-deployment configuration of the sensor network is not known a priori. It is unreasonable to use security algorithms that have strong dependence on locations of nodes in the network. For example, each sensor node u is expected to have only a few neighbours with which it can directly communicate. This is precisely the set of nodes with which u needs to share keys. However, the list cannot be determined before the actual deployment. An approximate knowledge of the locations of the nodes may strengthen the protocols, but robustness for handling run-time variations must be built in the protocols.

Mobility of sensor nodes Sensor nodes may be static or mobile. Mobile nodes change the network configurations (like the lists of neighbours) as functions of time and call for time-varying security tools.

Still, sensor nodes need to communicate secretly. The clear impracticality of using public-key routines forces one to use symmetric ciphers. But setting up symmetric keys among communicating nodes is a difficult task. The number n of nodes in a sensor network can range up to several hundred thousands. Storing a symmetric key for each pair of nodes is impossible, since that requires each sensor to have a memory large enough to store $n - 1$ keys. On the other extreme, every communication may use a single network-wide symmetric key. In that case the capture of a single node makes communication over the entire network completely insecure.

The plot thickens. There are graceful ways out. A host of algorithms has been recently proposed to address key establishment issues in sensor networks. In the rest of this appendix, we provide a quick survey of these tools. For the sake of simplicity, we assume here that our sensor network is static, that is, the nodes have no (or negligibly small) mobility. Though the schemes described below may be adapted to mobile networks, the required modifications are not necessarily easy and the current literature does not seem to be ready to take mobility into account.

We continue to deal with sensor processors of the capability of Atmel ATmega 128L. In practice, better processors (with speed, storage and cost roughly one order of magnitude higher) are available. We assume that the size (number of nodes) n of a sensor network is (usually) not bigger than a million, and also that a sensor node has of the order of 100 neighbours in its communication range.

B.3 The Basic Bootstrapping Framework

Key establishment in a sensor network is effected by a three-stage process called *bootstrapping*. Subsequent node-to-node communication uses the keys established during the bootstrapping phase. The three stages of bootstrapping are as follows:

Key predistribution This step is carried out before the deployment of the sensors. A key set-up server chooses a pool $\mathcal{K}$ of randomly generated keys and assigns to each sensor node u_i a subset $\mathcal{K}_i$ of $\mathcal{K}$. The set $\mathcal{K}_i$ is called the *key ring* of the node u_i. The key predistribution algorithms essentially differ in the ways the sets $\mathcal{K}$ and $\mathcal{K}_i$ are selected. Each key $k \in \mathcal{K}$ is associated with an ID that need not be kept secret and can even be transmitted in plaintext. Similarly, each sensor node is given a unique ID which need not be maintained secretly.

Direct key establishment Immediately after deployment, each sensor node tries to determine all other sensor nodes with which it can communicate directly and secretly. Two nodes that are within the communication ranges of one another are called *physical neighbours*, whereas two nodes sharing one (or more) key(s) in their key rings are called *key neighbours*. Two nodes can secretly (and directly) communicate with one another if and only if they are both physical and key neighbours; let us plan to call such pairs *direct neighbours*.

In the direct key establishment phase, each sensor node u locates its direct neighbours. To that end u broadcasts its own ID and the IDs of the keys in its key ring. Each physical neighbour v of u responds by mentioning the matching key IDs, if any, stored in the key ring of v. This is how u identifies its direct neighbours.

If sending unencrypted key IDs can be a potential threat to the security of the network, each node u can encrypt some plaintext message m by the keys in its ring and broadcasts the corresponding ciphertexts instead of the key IDs. Those physical neighbours of u that can decrypt one of the transmitted ciphertexts using one of the keys in their respective key rings establish themselves as direct neighbours of u.

Path key establishment This is an optional stage and, if executed, adds to the connectivity of the network. Suppose that two physical neighbours u and v fail to establish a direct link between them in the direct key establishment phase. But there exists a path $u = u_0, u_1, u_2, \ldots, u_{h-1}, u_h = v$ in the network with each u_i a direct neighbour of u_{i+1} (for $i = 0, 1, \ldots, h-1$). The node u then generates a random key k, encrypts k with the key shared between u and u_1 and sends the encrypted key to u_1. Subsequently, u_1 retrieves k by decryption, encrypts k by the key shared by u_1 and u_2 and sends this encrypted version of k to u_2. This process is repeated until the key k reaches the desired destination v. Now, u and v can communicate secretly and directly using k and thereby become direct neighbours.

The main difficulty in this process is the discovery of a path between u and v. This can be achieved by u initiating a message reflecting its desire to communicate with v. Let u_1 be a direct neighbour of u. If u_1 is also a direct neighbour of v, a path between u and v is discovered. Else u_1 retransmits u's request to the direct neighbours u_2 of u_1. This process is repeated, until a path is established between u and v, or the number of hops exceeds a certain limit. Note that path discovery may incur substantial

communication overhead and so the maximum number h of hops allowed needs to be fixed at a not-so-big value. Typically, the values $h = 2, 3$ are recommended.

A bootstrapping algorithm, or more precisely, a key predistribution algorithm must fulfill the following requirements. These requirements often turn out to be mutually contradictory. A key predistribution scheme attempts to achieve suitable trade-offs among them.

Compactness Each key ring should be small enough to fit in a sensor node's memory. Typically 50–200 cryptographic keys (say, 128-bit keys of block ciphers) can be stored in each processor. That number is between $n - 1$ (a key for each pair) and 1 (a master key for the entire network).

Randomness The key rings in different nodes are to be randomly chosen from a big pool, so that there is not enough overlap between the rings of two nodes.

Network connectivity The resulting network should be connected in the sense that the undirected graph $G = (V, E)$ with V comprising the nodes in the network and E containing a link (u, v) if and only if u and v are direct neighbours, must be (or at least with high probability) connected.

Resilience against node capture Ideally, the capture of any number of nodes must not divulge the secret key(s) between uncaptured direct neighbours. Practically, the fraction of communication links among uncaptured nodes, that are compromised because of node captures, must be small, at least as long as the fraction of nodes that are captured is not too high.

Scalability Arbitrarily (but not impractically) big networks should be supported.

Future addition of nodes One should allow new nodes to join the network at any point of time after the initial deployment, for example, to replenish captured, faulty and dead nodes.

Additional requirements may also be conceived of in order to take curative measures against active attacks and/or faults. However, a study of active attacks and of countermeasures against those is beyond the scope of our treatment here.

Detection of bad nodes There should be a mechanism to detect the presence and identities of dead, malfunctioning and rogue nodes. Here, a rogue node stands for a captured node that is used by the enemy to disrupt the natural working of the network. Active attacks mountable by the enemy include transmission of unauthorized and misleading data across the network, making neighbours always busy and letting them run out of battery sooner than the expected lifetime (*sleep deprivation attack*), and so on.

Revocation of bad nodes Faulty and rogue nodes must be pruned out of the network before they can cause sizeable harm.

Resilience against node replication Captured nodes can be replicated and the copies deployed by the enemy with the intention that these added nodes outnumber the legitimate nodes and eventually take control of the network. There should be a strategy to detect and cure replication of malicious nodes.

We now concentrate on some concrete realizations of the bootstrapping scheme. The optional third stage (path key establishment) will often be excluded from our discussion, because there are little algorithm-specific issues in this stage.

Before we introduce specific algorithms, let us summarize the notations we are going to use in the rest of this chapter:

n = Number of nodes in the sensor network
n' = (Expected) number of nodes in the physical neighbourhood of each node
d = Degree of connectivity of each node in the key/direct neighbourhood graph
P_{c} = Global connectivity (a high probability like 0.9999)
p' = Local connectivity (probability that two physical neighbours share a key)
M = Size of the key pool
m = The size of key ring of each node (in number of cryptographic keys)
$\mathbb{F}_q$ = The underlying field for the poly-pool and the matrix-pool schemes
S = Size of the polynomial (or matrix) pool
s = Number of polynomial (or matrix) shares in the key ring of each node
t = Degree of a polynomial (or dimension of a matrix)
c = Number of nodes captured
P_{e} = Probability of successful eavesdropping expressed as a function of c

B.4 The Basic Random Key Predistribution Scheme

The paper [88] by Eschenauer and Gligor is a pioneering research on bootstrapping in sensor networks. Their scheme, henceforth referred to as the *EG scheme*, is essentially the basic bootstrapping method just described.

The key set-up server starts with a pool $\mathcal{K}$ of randomly generated keys. The number M of keys in $\mathcal{K}$ is taken to be a small multiple of the network size n. For each sensor node u to be deployed, a random subset of m keys from $\mathcal{K}$ is selected and given to u as its key ring. Upon deployment, each node discovers its direct neighbours as specified in the generic description. We now explain how the parameters M, m are to be chosen so as to make the resulting network connected with high probability.

Let us first look at the key neighbourhood graph G_{key} on the n sensor nodes in which a link exists between two nodes if and only if these nodes are key neighbours. Let p denote the probability that a link exists between two randomly selected nodes of this graph. A result on random graphs due to Erdös and Rényi indicates that in the limit $n \to \infty$, the probability that G_{key} is connected is

$$P_{\mathrm{c}} = e^{-e^{-\gamma}}, \text{ where } p = \frac{\ln n}{n} + \frac{\gamma}{n}. \qquad \textbf{(B.1)}$$

We fix P_{c} at a high value, say, 0.9999, and express the expected degree of each node in G_{key} as

$$d = p(n-1) = \left(\frac{n-1}{n}\right)\left[\ln n - \ln(-\ln P_{\mathrm{c}})\right]. \qquad \textbf{(B.2)}$$

In practice, we should also bring physical neighbourhood in consideration and look at the direct neighbourhood graph $G = G_{\text{direct}}$ on the n deployed sensor nodes. In this graph, two nodes are connected by an edge if and only if they are direct neighbours. G is not random, since it depends on the geographical distribution of the nodes in the deployment area. However, we assume that the above result for random graphs continues to hold for G too. In particular, we fix the degree of direct connectivity of each node to be (at least) d and require

$$d = n'p', \tag{B.3}$$

where n' denotes the expected number of physical neighbours of each node, and where p' is the probability that two physical neighbours share one or more keys in their key rings $\mathcal{K}_i$ and $\mathcal{K}_j$. (P_{c} is often called the *global connectivity* and p' the *local connectivity*.)

For the determination of p', we first note that there is a total of $\binom{M}{m}$ key rings of size m that can be chosen from the pool $\mathcal{K}$ of size M. For a fixed $\mathcal{K}_i$, the total number of ways of choosing $\mathcal{K}_j$ such that $\mathcal{K}_j$ does *not* share a key with $\mathcal{K}_i$ is equal to the number of ways of choosing m keys from $\mathcal{K} \setminus \mathcal{K}_i$. This number is $\binom{M-m}{m}$. It then follows that

$$p' = 1 - \frac{\binom{M-m}{m}}{\binom{M}{m}} = 1 - \frac{((M-m)!)^2}{M!(M-2m)!} = 1 - \prod_{r=0}^{m-1} \frac{M-m-r}{M-r}\,. \tag{B.4}$$

Equations (B.2), (B.3) and (B.4) dictate how the key-pool size M is to be chosen, given the values of n, n' and m.

Example B.1 As a specific numerical example, consider a sensor network with $n = 10{,}000$ nodes. For the desired probability $P_{\text{c}} = 0.9999$ of connectedness of G_{key}, we use Equation (B.2) to obtain the desired degree d as $d \geqslant 18.419$. Let us take $d = 20$. Now, suppose that the expected number of physical neighbours of each deployed node is $n' = 50$. By Equation (B.3), we then require $p' = d/n' = 0.4$. Finally, assume that each sensor can hold $m = 150$ keys in its memory. Equation (B.4) indicates that we should have $M \leqslant 44{,}195$ in order to ensure $p' \geqslant 0.4$. In particular, we may take $M = 40{,}000$. □

Let us now study the resilience of the EG scheme against node captures. Assume that c nodes are captured at random from the network and that u and v are two uncaptured nodes that are direct neighbours. We compute the probability P_{e} that an eavesdropper can decipher encrypted communication between u and v based on the knowledge of the keys available from the c captured key rings. Clearly, smaller values of P_{e} indicate higher resilience against node captures.

Suppose that u and v use the key k for communication between them. Then, P_{e} is equal to the probability that k resides in one of the key rings of c captured nodes. Since each key ring consists of m keys randomly chosen from a pool of M keys, the probability that a particular key k is *not* available in a key ring is $\frac{M-m}{M} = 1 - \frac{m}{M}$ and consequently the probability that k does not appear in all of the c compromised key rings is $\left(1 - \frac{m}{M}\right)^c$. Thus, the probability of successful eavesdropping is

$$P_{\text{e}} = 1 - \left(1 - \frac{m}{M}\right)^c\,.$$

Example B.2 As in Example B.1, take $n = 10{,}000$, $n' = 50$, $m = 150$ and $M = 40{,}000$. If $c = 100$ nodes are captured, the fraction of compromised communication is $P_e \approx 0.313$. Thus, a capture of only 100 nodes leads to a compromise of about one-third of the traffic. That is not a satisfactory figure. We need better algorithms. □

B.4.1 The q-composite Scheme

Chan et al. [44] propose several modifications of the basic EG scheme in order to improve upon the resilience of the network against node capture. The *q-composite scheme*, henceforth abbreviated as the *qC scheme*, is based on the requirement of a bigger overlap of key rings for enabling nodes to communicate.

As in the EG scheme, the key set-up server decides a pool $\mathcal{K}$ of M random keys and loads the key ring of each node with a random subset of size m of $\mathcal{K}$. Let the network consist of n nodes.

In the direct key establishment phase, each node u discovers all its physical neighbours that share q or more keys with u, where q is a predetermined system-wide parameter. Those physical neighbours that do so are now called direct neighbours of u. Let v be a direct neighbour of u and let $q' \geqslant q$ be the actual number of keys shared by u and v. Call these keys $k_1, k_2, \ldots, k_{q'}$. The nodes use the key

$$k := H(k_1||k_2||\cdots||k_{q'})$$

for future communication, where $||$ denotes string concatenation and H is a hash function. A pair of physical neighbours that share $< q$ predistributed keys do not communicate directly.

Recall that for the basic EG scheme $q = 1$ and the key k for communication between direct neighbours is taken to be one shared key instead of a hash value of all shared keys. The motivation behind going for the qC scheme is that requiring a bigger overlap between the key rings of a pair of physical neighbours leads to a smaller probability P_e of successful eavesdropping, since now the eavesdropper has to possess the knowledge of at least q shared keys (not just one). However, the requirement of q (or more) matching keys between communicating nodes restricts the key pool size M more than the EG scheme, and consequently a capture of fewer nodes reveals a bigger fraction of the total key pool $\mathcal{K}$ to the eavesdropper. Chan et al. [44] report that the best trade-off is achieved for the value $q = 2$ or 3.

Let us now derive the explicit expressions for M and P_e. Equations (B.1), (B.2) and (B.3) hold for the qC scheme with the sole exception that now the interpretation of the probability p' of direct neighbourhood is different. There is a total of $\binom{M}{m}^2$ ways of choosing two random key rings of size m from a pool of M keys. Let us compute the number p'_r of such pairs of key rings sharing *exactly* r keys. First, these shared r keys can be chosen in $\binom{M}{r}$ ways. Out of the remaining $M - r$ keys, the remaining $m - r$ keys for the first ring can be chosen in $\binom{M-r}{m-r}$ ways. Finally, the remaining $m - r$ keys for the second ring can be chosen in $\binom{M-m}{m-r}$ ways from $M - m$ keys not present in the first ring. Thus, we have

$$p'_r = \frac{\binom{M}{r}\binom{M-r}{m-r}\binom{M-m}{m-r}}{\binom{M}{m}^2},$$

that is,

$$p' = 1 - (p'_0 + p'_1 + \cdots + p'_{q-1}) = p'_q + p'_{q+1} + \cdots + p'_m$$

is the equivalent of Equation (B.4) for the qC scheme.

Example B.3 As in Example B.1, consider $n = 10{,}000$, $n' = 50$, $m = 150$. For $d = 20$, we require $p' \geqslant 0.4$. This, in turn, demands $M \leqslant 16{,}387$ for $q = 2$ and $M \leqslant 9{,}864$ for $q = 3$. Compare these with the requirement $M \leqslant 44{,}195$ for the EG scheme. □

Let us now calculate the probability P_{e} of successfully deciphering the communication between two uncaptured nodes u and v, given that c nodes are already captured by the eavesdropper. Let $q' \geqslant q$ be the actual number of keys shared by u and v; this happens with probability $p'_{q'}/p'$. Each of these common keys is available to the eavesdropper with a probability $1 - \left(1 - \frac{m}{M}\right)^c$. It follows that

$$P_{\mathrm{e}} = \sum_{q'=q}^{m} \left[1 - \left(1 - \frac{m}{M}\right)^c\right]^{q'} \left(\frac{p'_{q'}}{p'}\right).$$

Example B.4 Let us continue with the network of Examples B.1, B.2 and B.3. The following table summarizes the probabilities P_{e} for various values of c. For the EG scheme, we take $M = 40{,}000$, whereas for the qC scheme, we take $M = 16{,}000$ for $q = 2$ and $M = 9{,}800$ for $q = 3$.

	P_{e}							
Scheme	$c = 10$	$c = 20$	$c = 30$	$c = 40$	$c = 50$	$c = 75$	$c = 100$	$c = 150$
EG	0.037	0.072	0.107	0.140	0.171	0.246	0.313	0.431
2C	0.005	0.019	0.041	0.068	0.101	0.196	0.300	0.499
3C	0.002	0.011	0.032	0.066	0.111	0.255	0.413	0.678

This table indicates that when the number of nodes captured is small, the qC scheme outperforms the EG scheme. However, for large values of c, the effects of smaller values of the key-pool size show up, leading to a poorer performance of the qC schemes compared to the EG scheme. □

B.4.2 Multi-path Key Reinforcement

Another way to improve the resilience of the network against node captures is the multi-path key reinforcement scheme proposed again by Chan et al. [44]. As in the EG scheme, sensor nodes are deployed each with m keys in its key ring chosen randomly from a pool of M keys. Let u and v establish themselves as direct neighbours sharing the key k. Instead of using k itself as the key for future communication, the nodes try to locate several pairwise node-disjoint paths between them. Such a path $u = v_0, v_1, \ldots, v_l = v$ consists of pairs of direct neighbours (v_i, v_{i+1}) for $i = 0, \ldots, l-1$. A randomly generated key k' is then routed securely along the path from u to v.

Assume that r node-disjoint paths between u and v are discovered and the random keys $k'_1, k'_2, \ldots, k'_r$ are transfered securely along these paths. The nodes u and v then use the key

$$k' := k \oplus k'_1 \oplus \cdots \oplus k'_r$$

for future communication.

The reason why this scheme improves resilience against node captures is that even if the original k resides in the memory of a captured node, the new key k' is computable by the adversary if and only if she can obtain all of the r session secrets $k'_1, \ldots, k'_r$. The bigger r is, the more difficult it is for the adversary to eavesdrop on all of the r node-disjoint paths. On the other hand, if the lengths of these paths are large, then the probability of eavesdropping at some links of the paths increases. Moreover, increasing the lengths of the paths incurs bigger communication overhead. The proponents of the scheme recommend only 2-hop multi-path key reinforcement.

We do not go into the details of the analysis of the multi-path key reinforcement scheme, but refer the reader to Chan et al. [44]. We only note that though it is possible to use multi-path key reinforcement for the q-composite scheme, it is not a lucrative option. The smaller size of the key pool for the q-composite scheme tends to nullify the effects of multi-path key reinforcement.

B.5 Random Pairwise Scheme

A *pairwise key predistribution scheme* offers perfect resilience against node captures, that is, the capture of any number c of nodes does not reveal any information about the secrets used by uncaptured nodes. This corresponds to $P_{\mathrm{e}} = 0$ irrespective of c. This desirable property of the network is achieved by giving each key to the key rings of only two nodes. Moreover, the sharing of a key k between two unique nodes u and v implies that these nodes can *authenticate* themselves to one another — no other node possesses k and can prove itself as u to v or as v to u.

Pairwise keys can be distributed to nodes in many ways. Now, we deal with random distribution. Let m denote the size of the key ring of each sensor node. For each node u in the network, the key set-up server randomly selects m other nodes $v_1, \ldots, v_m$ and distributes a new random key k_i to each of the pairs (u, v_i) for $i = 1, \ldots, m$. This distribution mechanism should also ensure that two nodes u, v in the network share at most one key. If k is given to u and v, the set-up server also attaches the ID of v to the copy of k in the key ring of u and the ID of u to the copy of k in the key ring of v.

In the direct key establishment phase, each node u broadcasts its own ID. Each physical neighbour v of u, that finds the ID of u stored against a key in the key ring of v, identifies u as its direct neighbour and also the unique key shared by u and v.

The analysis of the random pairwise scheme is a bit tricky. Here, the global connectivity graph G_{key} is m-regular, that is, each node has degree exactly m and we cannot expect to maintain this degree locally too. On the other hand, it is reasonable to assume under a random deployment model that the fraction of nodes with which a given node shares pairwise keys remains the same both locally and globally. More precisely, we equate p' with p, that is,

$$\frac{d}{n'} = p' = p = \frac{m}{n-1} \approx \frac{m}{n}. \qquad \textbf{(B.5)}$$

Here, d denotes the desired local degree of a node. Equation (B.2) gives the formula for d in terms of the global connectivity P_{c}. For $P_{\mathrm{c}} = 0.9999$, we have $d = 16.11$ for

$n = 1{,}000$, $d = 18.42$ for $n = 10{,}000$, $d = 20.72$ for $n = 100{,}000$, and $d = 23.03$ for $n = 1{,}000{,}000$. That is, the value of d does not depend heavily on n, as long as n ranges over practical values. In particular, one may fix $d = 20$ (or $d = 25$ more conservatively) for all applications.

Equation (B.5) implies

$$n \approx \frac{mn'}{d}\,.$$

This equation reflects the drawback of the random pairwise scheme. The value m is limited by the memory of a sensor node, whereas n' is dictated by the density of nodes in the deployment area and d can be taken as a constant, and so the network size n is bounded above by the quantity $\frac{mn'}{d}$ called the *maximum supportable network size*. The basic scheme (and its variants) support networks of arbitrarily large sizes, whereas the random pairwise scheme has only limited supports.

Example B.5 Take $m = 150$, $n' = 50$ and $d = 20$. The maximum supportable network size is then $\frac{150 \times 50}{20} = 375$. This is too small to be useful. We require modifications of the random pairwise scheme in order to be able to use it in practice. □

B.5.1 Multi-hop Range Extension

Since m and d are limited by hard constraints, the only way to increase the maximum supportable network size is to increase the *effective* size n' of the physical neighbourhood of a node. The multi-hop range extension strategy accomplishes that. In the direct key establishment phase, each node u broadcasts its ID. Each physical neighbour v of u re-broadcasts the ID of u. Each physical neighbour w of v then re-re-broadcasts the ID of u. This process is continued for a predetermined number r of hops. Any node u' reachable from u in $\leqslant r$ hops and sharing a pairwise key with u can now establish a path of secure communication with u. During a future communication between u and u', the intermediate nodes in the path simply forward a message encrypted by the pairwise key between u and u'. Using r hops thereby increases the effective radius of physical neighbourhood by a factor of r, and consequently the number of effective neighbours of each node gets multiplied by a factor of r^2. Thus, the maximum supportable network size now becomes

$$n \approx \frac{mn'r^2}{d}.$$

For $r = 3$ and for the parameters of Example B.5, this size now attains a more decent value of 3375.

Increasing r incurs some cost. First, the communication overhead increases quadratically with r. Second, since intermediate nodes in a multi-hop path simply retransmit messages without authentication, chances of specific active attacks at these nodes increase. Large values of r are, therefore, discouraged.

B.6 Polynomial-pool-based Key Predistribution

Liu and Ning's *polynomial-pool-based key predistribution scheme* (abbreviated as the *poly-pool scheme*) [181, 183] is based on the idea presented by Blundo et al. [28]. Let $\mathbb{F}_q$ be a finite field with q just large enough to accommodate a symmetric encryption key. For a 128-bit block cipher, one may take q to be smallest prime larger than 2^{128} (prime field) or 2^{128} itself (extension field of characteristic 2). Let $f(X,Y) \in \mathbb{F}_q[X,Y]$ be a bivariate polynomial that is assumed to be symmetric, that is, $f(X,Y) = f(Y,X)$. Let t be the degree of f in each of X and Y. A *polynomial share* of f is a univariate polynomial $f^{(\alpha)}(X) := f(X,\alpha)$ for some element $\alpha \in \mathbb{F}_q$. Two shares $f^{(\alpha)}$ and $f^{(\beta)}$ of the same polynomial f satisfy

$$f^{(\alpha)}(\beta) = f(\beta,\alpha) = f(\alpha,\beta) = f^{(\beta)}(\alpha)\,. \tag{B.6}$$

Thus, if the shares $f^{(\alpha)}$, $f^{(\beta)}$ are given to two nodes, they can come up with the common value $f(\alpha,\beta) \in \mathbb{F}_q$ as a shared secret between them.

Given $t+1$ or more shares of f, one can reconstruct $f(X,Y)$ uniquely using Lagrange's interpolation formula (Exercise 2.53). On the other hand, if only t or less shares are available, there are many (at least q) possibilities for f and it is impossible to determine f uniquely. So the disclosure of up to t shares does not reveal the polynomial f to an adversary and uncompromised shared keys based on f remain secure.

Using a single polynomial for the entire network is not a good proposal, since t is limited by memory constraints in a sensor node. In order to increase resilience against node captures, many bivariate polynomials need to be used, and shares of random subsets of this polynomial pool are assigned to the key rings of individual nodes. This is how the poly-pool scheme works. If the degree t equals 0, this scheme degenerates to the EG scheme.

The key set-up server first selects a random pool $\mathcal{F}$ of S symmetric bivariate polynomials in $\mathbb{F}_q[X,Y]$ each of degree t in X and Y. Some IDs $\alpha_1, \alpha_2, \ldots, \alpha_n \in \mathbb{F}_q$ are also generated for the nodes in the network. For each node u in the network, s polynomials $f_1, f_2, \ldots, f_s$ are randomly picked up from $\mathcal{F}$ and the polynomial shares $f_1(X,\alpha), f_2(X,\alpha), \ldots, f_s(X,\alpha)$ are loaded in the key ring of u, where α is the ID of u. Each key ring now requires space for storing $s(t+1)\log q$ bits, that is, for storing $m := s(t+1)$ symmetric keys.

Upon deployment, each node u broadcasts the IDs of the polynomials, the shares of which reside in its key ring. Each physical neighbour v of u, that has shares of some common polynomial(s), establishes itself as a direct neighbour of u. The exact pairwise key k between u and v is then calculated using Equation (B.6). If broadcasting polynomial IDs in plaintext is too unsafe, each node u can send some message encrypted by potential pairwise keys based on its polynomial shares. Those physical neighbours that can decrypt one of these encrypted messages have shares of common polynomials.

Like the EG scheme, the poly-pool scheme can be analysed under the framework of random graphs. Equations (B.1), (B.2) and (B.3) continue to hold under the poly-pool scheme. However, in this case the local connection probability p' is computed as

$$p' \;=\; 1 - \frac{\binom{S-s}{s}}{\binom{S}{s}} \;=\; 1 - \frac{((S-s)!)^2}{S!(S-2s)!} \;=\; 1 - \prod_{r=0}^{s-1} \frac{S-s-r}{S-r}\,. \tag{B.7}$$

Given constraints on the network and the nodes, the desired size S of the polynomial pool $\mathcal{F}$ can be determined from this formula.

Let us now compute the probability P_{e} of compromise of communication between two uncaptured nodes u, v as a function of the number c of captured nodes. If $c \leqslant t$, the eavesdropper cannot gather enough polynomial shares to learn about any polynomial in $\mathcal{F}$, that is, $P_{\mathrm{e}} = 0$. So assume that $c > t$ and let p_r denote the probability that exactly r shares of a given polynomial f (say, the one whose shares are used by the two uncaptured nodes u, v) are available in the key rings of the c captured nodes. The probability that a share of f is present in a key ring is $\frac{s}{S}$ and so (by the Bernoulli distribution)

$$p_r = \binom{c}{r} \left(\frac{s}{S}\right)^r \left(1 - \frac{s}{S}\right)^{c-r}. \tag{B.8}$$

Since $t + 1$ or more shares of f are required for the determination of f, we have

$$P_{\mathrm{e}} = 1 - (p_0 + p_1 + \cdots + p_t) = p_{t+1} + p_{t+2} + \cdots + p_c\,. \tag{B.9}$$

Example B.6 Let $n = 10{,}000$ (network size), $n' = 50$ (expected size of physical neighbourhood of a node), $m = 150$ (key ring size in number of symmetric keys) and $P_{\mathrm{c}} = 0.9999$ (global connectivity). Let us plan to choose bivariate polynomials of degree $t = 49$, so that each key ring can hold $s = 3$ polynomial shares.

For the determination of S, we first compute $d = 20$ as in Example B.1. We then require $p' \geqslant \frac{d}{n'} = 0.4$. The biggest size S satisfying this bound is derived from Equation (B.7) as $S = 20$.

The following table lists the probability P_{e} for various values of c.

c	50	100	150	200	250	300	350	400
P_{e}	6.38×10^{-42}	2.30×10^{-16}	1.70×10^{-8}	1.52×10^{-4}	0.0196	0.231	0.668	0.932

The table shows substantial improvement in resilience against node capture as achieved by the poly-pool scheme over the EG and qC schemes. □

B.6.1 Pairwise Key Predistribution

The poly-pool scheme can be made pairwise by allowing no more than $t + 1$ shares of any polynomial to be distributed among the nodes. The best that the adversary can achieve is a capture of nodes with all these $t + 1$ shares and a subsequent determination of the corresponding bivariate polynomial. But this knowledge does not help the adversary, since no other node in the network uses a share of this compromised polynomial. That is, two uncaptured nodes continue to communicate with perfect secrecy.

However, like the random pairwise scheme, the pairwise poly-pool scheme suffers from the drawback that the *maximum supportable network size* is now limited by the quantity $\frac{(t+1)S}{s}$. For the parameters of Example B.6, this size turns out to be an impractically low 333.

B.6.2 Grid-based Key Predistribution

The grid-based key predistribution considerably enhances the resilience of the network against node captures. To start with, let us play a bit with Example B.6.

Example B.7 Take $n = 10{,}000$, $n' = 50$ and $m = 150$. We calculated that the optimal value of S that keeps the network connected with high probability is $S = 20$. Now, let us instead take a much bigger value of S, say, $S = 200$. First, let us look at the brighter side of this choice. The probability P_{e} is listed in the following table as a function of c .

c	500	1000	1500	2000	2500	3000	3500	4000
P_{e}	1.90×10^{-25}	4.88×10^{-13}	3.10×10^{-7}	4.68×10^{-4}	0.0282	0.245	0.655	0.917

That is a dramatic improvement in the resilience figures. It, however, comes at a cost. The optimal value $S = 20$ was selected in Example B.6 in order to achieve a desired connectivity in the network. With $S = 200$, the probability p' reduces from 0.404 to $1 - \binom{197}{3}/\binom{200}{3} = 0.0445$, and each node is expected to have only about 2 direct neighbours. As a result, the network is likely to remain *disconnected* with high probability. □

The grid-based key predistribution allocates polynomial shares cleverly to the nodes so as to achieve resilience figures of the last example with a reasonable guarantee that the resulting network remains connected. Let n be the size of the network and take $\sigma := \lceil \sqrt{n} \rceil$. For the sake of simplicity, let us assume that $n = \sigma^2$. The n nodes are then placed on a $\sigma \times \sigma$ square grid. The node at the (i, j)-th grid location (where $i, j \in \{1, 2, \ldots, \sigma\}$) is identified by the pair (i, j). The set-up server generates 2σ random symmetric bivariate polynomials $f_i^{(r)}(X, Y), f_j^{(c)}(X, Y) \in \mathbb{F}_q[X, Y]$, $1 \leqslant i, j \leqslant \sigma$, each of degree t in both X and Y. The i-th polynomial $f_i^{(r)}$ corresponds to the i-th row and the j-th polynomial $f_j^{(c)}$ to the j-th column in the grid. The key ring of the node at location (i, j) in the grid is given the two polynomial shares $f_i^{(r)}(X, j)$ and $f_j^{(c)}(X, i)$. The memory required for this is equivalent to the storage for $2(t + 1)$ symmetric keys.

Now, look at the key establishment phase. Let two nodes u, v with IDs (i, j) and (i', j') be physical neighbours after deployment. First, consider the simple case $i = i'$. Both the nodes have shares of the polynomial $f_i^{(r)}$ and can arrive at the common secret value $f_i^{(r)}(j, j') = f_i^{(r)}(j', j)$ using the column identities of one another. Similarly, if $j = j'$, the nodes can compute the shared secret $f_j^{(c)}(i, i') = f_j^{(c)}(i', i)$. It follows that each node can establish keys directly with $2(\sigma - 1)$ other nodes in the network. That's, however, a truly small fraction of the entire network.

Assume now that $i \neq i'$ and $j \neq j'$. If the node w with identity either (i, j') or (i', j) is in the physical neighbourhood of both u and v, then there is a secure link between u and w, and also one between w and v. The nodes u and v can then establish a path key via the intermediate node w.

So suppose also that neither (i, j') nor (i', j) resides in the communication ranges of both u and v. Consider the nodes $w_1 := (i, k)$ and $w_2 := (i', k)$ for some $k \in \{1, 2, \ldots, \sigma\} \setminus \{j, j'\}$. Suppose further that w_1 is in the physical neighbourhood of u, w_2 in that of w_1 and v in that of w_2. But then there is a secure u, v-path comprising the links $u \to w_1$, $w_1 \to w_2$ and $w_2 \to v$. Similarly, the nodes (k, j) and (k, j') for each

$k \neq i, i'$ can help u and v establish a path key. To sum up, there are $2(\sigma - 2)$ potential three-hop paths between u and v.

If all these three-hop paths fail, one may go for four-hop, five-hop, ... paths, but at the cost of increased communication overhead. As argued in Liu and Ning [181, 183], exploring paths with $\leqslant 3$ hops is expected to give the network high connectivity.

For the grid-based scheme, we have $S = 2\sigma$ (the key pool size) and $s = 2$ (the number of polynomial shares in each node's key ring). Thus, the probability P_{e} can now be derived like Equations (B.8) and (B.9) as

$$P_{\mathrm{e}} = 1 - (p_0 + p_1 + \cdots + p_t) = p_{t+1} + p_{t+2} + \cdots + p_c,$$

where

$$p_r = \binom{c}{r} \frac{(\sigma - 1)^{c-r}}{\sigma^c}.$$

Example B.8 Take $n = 10{,}000$ and $m = 150$. Since each node has to store only two polynomial shares, we now take $t = 74$. Moreover, $\sigma = 100$, that is, the size of the polynomial pool is $S = 200$. The probability P_{e} can now be tabulated as a function of c (number of nodes captured) as follows:

c	1000	2000	3000	4000	5000	6000	7000
P_{e}	2.45×10^{-40}	1.99×10^{-21}	2.68×10^{-12}	4.35×10^{-7}	5.41×10^{-4}	0.0334	0.290

This is a very pretty performance. The capture of even 60 per cent of the nodes leads to a compromise of only 3.34 per cent of the communication among uncaptured nodes. □

This robustness of the grid-based distribution comes at a cost, though. The path key establishment stage is communication-intensive and is mandatory for ensuring good connectivity. Moreover, this stage is based on the assumption that during bootstrapping not many nodes are captured. If this assumption cannot necessarily be enforced, the scheme forfeits much of its expected resilience guarantees.

B.7 Matrix-based Key Predistribution

The matrix-based key predistribution scheme is derived from the idea proposed by Blom [25]. It is similar to the polynomial-based key predistribution and employs symmetric matrices (in place of symmetric polynomials). Let $\mathbb{F}_q$ be a finite field with q just large enough to accommodate a symmetric key and let G be a $t \times n$ matrix over $\mathbb{F}_q$, where t is determined by the memory of a sensor node and n is the number of nodes in the network. It is *not* required to preserve G with secrecy. Anybody, even the enemies, may know G. We only require G to have rank t, that is, any t columns of G must be linearly independent. If g is a primitive element of $\mathbb{F}_q^*$, the following matrix is recommended.

$$G = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ g & g^2 & g^3 & \cdots & g^n \\ g^2 & g^4 & g^6 & \cdots & g^{2n} \\ g^3 & g^6 & g^9 & \cdots & g^{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ g^{t-1} & g^{2(t-1)} & g^{3(t-1)} & \cdots & g^{n(t-1)} \end{pmatrix}. \quad \textbf{(B.10)}$$

In a memory-starved environment, this G has a compact representation, since its j-th column is uniquely identified by the value g^j. The remaining elements in the column can be easily computed by performing few multiplications.

Let D be a *secret* $t \times t$ *symmetric* matrix, and A the $n \times t$ matrix defined by:

$$A := (DG)^{\mathrm{t}} = G^{\mathrm{t}} D^{\mathrm{t}} = G^{\mathrm{t}} D.$$

Finally, define the $n \times n$ matrix

$$K := AG.$$

It follows that $K = AG = G^{\mathrm{t}} DG = G^{\mathrm{t}} (G^{\mathrm{t}} D^{\mathrm{t}})^{\mathrm{t}} = G^{\mathrm{t}} (G^{\mathrm{t}} D)^{\mathrm{t}} = G^{\mathrm{t}} A^{\mathrm{t}} = (AG)^{\mathrm{t}} = K^{\mathrm{t}}$, that is, K is a symmetric matrix. If the (i, j)-th element of K is denoted by k_{ij}, we have $k_{ij} = k_{ji}$, that is, this common value can be used as a pairwise key between the i-th and j-th nodes.

Let the (i, j)-th element of A be denoted by a_{ij} for $1 \leqslant i \leqslant n$ and $1 \leqslant j \leqslant t$. Also let g_{ij}, $1 \leqslant i \leqslant t$ and $1 \leqslant j \leqslant n$, denote the (i, j)-th element of G. But then the pairwise key $k_{ij} = k_{ji}$ is expressed as:

$$k_{ij} = \sum_{l=1}^{t} a_{il} g_{lj} = \sum_{l=1}^{t} a_{jl} g_{li} = k_{ji}.$$

Thus, the i-th row of A and the j-th column of G suffice for the i-th node to compute k_{ij}. Similarly, the j-th row of A and the i-th column of G allow the j-th node to compute k_{ji}. In view of this, every node, say, the i-th node, is required to store the i-th row of A and the i-th column of G. If G is as in Equation (B.10), only g^i needs to be stored instead of the full i-th column of G. Thus, the storage of $t + 1$ elements of $\mathbb{F}_q$ (equivalent to $t + 1$ symmetric keys) suffices.

During direct key establishment, two physical neighbours exchange their respective columns of G for the computation of the common key. Since G is allowed to be a public knowledge, this communication does not reveal secret information to the adversary.

Suppose that the adversary gains knowledge of some $t' \geqslant t$ rows of A (say, by capturing nodes). We also assume that the matrix G is completely known to the adversary. The adversary picks up any t known rows of A and constructs a $t \times t$ matrix A' comprising these rows. But then $A' = G'D$, where G' is a suitable $t \times t$ submatrix of G. Since G is assumed to be of rank t, G' is invertible and so the secret matrix D can be easily computed. Conversely, if D is known to the adversary, she can compute A and, in particular, any $t' \geqslant t$ rows of A.

If only $t' < t$ rows are known to the adversary, then any choice of any $t - t'$ unknown rows of A yields a value of the matrix D, and subsequently we can construct the remaining $n-t$ unknown rows of A. In other words, D cannot be uniquely recovered

from a knowledge of less than t rows of A. This task is difficult too, since there is an infeasible number of choices for assigning values to the elements of the unknown $t - t'$ rows of A.

To sum up, the matrix-based key predistribution scheme is completely secure, if less than t nodes are only captured. On the other hand, if t or more nodes are captured, then the system is completely compromised. Thus, the resilience against node capture of this scheme is determined solely by t and is independent of the size n of the network. The parameter t, in turn, is restricted by the memory of a sensor node (a node has to store $t + 1$ elements of $\mathbb{F}_q$).

In order to overcome this difficulty, Du et al. [79] propose a matrix-pool-based scheme. Here, S matrices $A_1, A_2, \ldots, A_S$ are computed from S pairwise different secret matrices $D_1, D_2, \ldots, D_S$. The same G may be used for all these key spaces. Each node is given shares (that is, rows) of s matrices randomly chosen from the pool $\{A_1, A_2, \ldots, A_S\}$. The resulting details of the matrix-pool-based scheme are quite analogous to those pertaining to the polynomial-pool-based scheme described in the earlier section, and are omitted here.

B.8 Location-aware Key Predistribution

The key predistribution algorithms discussed so far are based on a random deployment model. In practice, the deployment model (like the expected location of each node and the overall geometry of the deployment area) may be known a priori. This knowledge can be effectively exploited to tune the key predistribution algorithms so as to achieve better connectivity and higher resilience against node capture. As an example, consider sensor nodes deployed from airplanes in groups or scattered uniformly from trucks. Since the approximate tracks of these vehicles are planned a priori, the key rings of the nodes can be loaded appropriately to achieve the expected performance enhancements.

Two nodes that are in the physical neighbourhoods of one another need only share a pairwise key. Therefore, the basic objective of designing location-aware schemes is to predistribute keys in such a way that two nodes that are expected to remain close in the deployment area are given common pairwise keys, whereas two nodes that are expected to be far away after deployment need not share any pairwise key. The actual deployment locations of the nodes cannot usually be predicted accurately. Nonetheless, an approximate knowledge of the locations can boost the performance of the network considerably. The smaller the errors between the expected and actual locations of the nodes are, the better a location-aware scheme is expected to perform.

B.8.1 Closest Pairwise Keys Scheme

Liu and Ning [182] propose a modification of the random pairwise key scheme (Section B.5) based on deployment knowledge. Let there be n sensor nodes in the network with each node capable of storing m cryptographic keys. The expected deployment location of each node is provided to the key set-up server. For each node u in the network, the server determines m other nodes whose expected locations of deployment are closest to that of u and for which pairwise keys with u have not already been

established. For every such node v, a new random key k_{uv} is generated. The key-plus-ID combination (k_{uv}, v) is loaded in u's key ring, whereas the pair (k_{uv}, u) is loaded in v's key ring.

This natural and simple-minded strategy provides complete security against node captures, as it is a pairwise key distribution scheme. Now, there is no limitation on the maximum supportable network size (under the reasonable assumption that there are much less than 2^l nodes in the network, where l is the bit length of a cryptographic key, say, 64 or 128). Moreover, the incorporation of deployment knowledge increases the connectivity of the network. In order to analyse this gain, we first introduce some formal notations.

For the sake of simplicity, we assume that the deployment region is two-dimensional, so that every point in that region is expressed by two coordinates x and y. Let u be a sensor node whose expected deployment location is (u_x, u_y) and whose actual deployment location is (u'_x, u'_y). This corresponds to a deployment error of $e_u = (u'_x - u_x, u'_y - u_y)$. The actual location (u'_x, u'_y) (or equivalently the error e_u) is modelled as a continuous random variable that can assume values in $\mathbb{R}^2$. The probability density function f_u of (u'_x, u'_y) characterizes the pattern of deployment error. One possibility is to assume that (u'_x, u'_y) is uniformly distributed within a circle with centre at (u_x, u_y) and of radius ϵ called the *maximum deployment error*. We then have:

$$f_u(u'_x, u'_y) = \begin{cases} \frac{1}{\pi\epsilon^2} & \text{if } (u'_x - u_x)^2 + (u'_y - u_y)^2 \leqslant \epsilon^2, \\ 0 & \text{otherwise.} \end{cases} \qquad \textbf{(B.11)}$$

An arguably more realistic strategy is to model (u'_x, u'_y) as a random variable following the two-dimensional normal (Gaussian) distribution with mean (u_x, u_y) and variance σ^2. The corresponding density function is:

$$f_u(u'_x, u'_y) = \frac{1}{2\pi\sigma^2} e^{-\left[(u'_x - u_x)^2 + (u'_y - u_y)^2\right]/(2\sigma^2)}.$$

Let u and v be two deployed nodes. We assume that each node has a communication range of ρ. We also make the simplifying assumption that the different nodes are deployed independently, that is, (u'_x, u'_y) and (v'_x, v'_y) are independent random variables. The probability that u and v lie in the communication ranges of one another can be expressed as a function of the expected locations (u_x, u_y) and (v_x, v_y) as:

$$p(u, v) = p(u_x, u_y, v_x, v_y) = \int\int\int\int_C f_u(u'_x, u'_y) f_v(v'_x, v'_y)\, \mathrm{d}u'_x\, \mathrm{d}u'_y\, \mathrm{d}v'_x\, \mathrm{d}v'_y.$$

Here, the integral is over the region C of $\mathbb{R}^4$ defined by $(u'_x - v'_x)^2 + (u'_y - v'_y)^2 \leqslant \rho^2$.

Let n' denote the number of physical neighbours of u (or of any sensor node). We know that u shares pairwise keys with exactly m nodes. We assume that these key neighbours of u are distributed uniformly in a circle centred at u and of radius ρ'. The expected value of ρ' is:

$$\rho' = \rho \times \sqrt{\frac{m}{n'}}.$$

Let v be a key neighbour of u. The probability that v lies in the physical neighbourhood of u is given by

$$p(u) = p(u_x, u_y) = \frac{1}{\pi\rho'^2} \int\int_{C'} p(u_x, u_y, v_x, v_y)\, \mathrm{d}v_x\, \mathrm{d}v_y\,,$$

where C' is the region $(v_x - u_x)^2 + (v_y - u_y)^2 \leqslant \rho'^2$. Therefore, u is expected to have $m \times p(u)$ direct neighbours. Since the size of the physical neighbourhood of u is n', the local connectivity, that is, the probability that u can establish a pairwise key with a physical neighbour is given by

$$p' = p(u) \times \frac{m}{n'} .$$

In general, it is difficult to compute the above integrals. Liu and Ning [182] compute the probability p' for the density function given by Equation (B.11) and establish that $p' \approx 1$ for small deployment errors, namely $\epsilon \leqslant \rho$. As ϵ increases, p' gradually reduces to the corresponding probability for the random pairwise scheme.

In order to add sensor nodes at a later point of time, the key set-up server again uses deployment knowledge. The keys rings of the new nodes are loaded based on the expected deployment locations of these nodes and on the (expected or known) locations of the deployed nodes. Pairwise keys between the new and the deployed nodes are communicated to the deployed nodes over secure channels (routing through uncompromised nodes).

B.8.2 Location-aware Polynomial-pool-based Scheme

Several variants of the closest pairwise keys scheme have been proposed. Liu and Ning themselves propose an extension based on pseudorandom functions [182]. Du et al. propose a variant of the basic (EG) scheme based on a specific model of deployment [80]. We end this section by briefly outlining a location-aware adaptation of the polynomial-pool-based scheme (Section B.6).

For simplicity, let us assume that the deployment region is a rectangular area. This region is partitioned into a 2-dimensional array of rectangular cells. Let the partition consist of R rows and C columns. The cell located at the i-th row and the j-th column is denoted by $C_{i,j}$. The neighbours of the cell $C_{i,j}$ are taken to be the four adjacent cells: $C_{i-1,j}, C_{i+1,j}, C_{i,j-1}, C_{i,j+1}$.

The key set-up server first decides a finite field $\mathbb{F}_q$ with q just big enough to accommodate a cryptographic key. The server also chooses $R \times C$ random symmetric bivariate polynomials $f_{i,j}(X, Y) \in \mathbb{F}_q[X, Y]$, $1 \leqslant i \leqslant R$, $1 \leqslant j \leqslant C$. The polynomial $f_{i,j}$ is meant for the cell $C_{i,j}$. The degree t (in both X and Y) of each $f_{i,j}$ is so chosen that each sensor node has sufficient memory to store the shares of five such polynomials.

Let u be a node to be deployed and let the expected deployment location of u lie in the cell $C_{i,j}$ called the *home cell* of u. The key ring of u is loaded with the shares (evaluated at u) of the five polynomials corresponding to the home cell and its four neighbouring cells. More precisely, u gets the five shares: $f_{i,j}(X, u)$, $f_{i-1,j}(X, u)$, $f_{i+1,j}(X, u)$, $f_{i,j-1}(X, u)$, and $f_{i,j+1}(X, u)$. The set-up server also stores in u's memory the ID (i, j) of its home cell.

In the direct key establishment phase, each node u broadcasts the ID (i, j) of its home cell (or some messages encrypted by potential pairwise keys). Those physical neighbours whose home cells are either the same as or neighbouring to that of u can establish pairwise keys with u.

An analysis of the performance of this location-aware poly-pool-based scheme can be carried out along similar lines to the closest pairwise scheme. We leave out the details here and refer the reader to Liu and Ning [182].

This page is intentionally left blank

Complexity Theory and Cryptography

C

C.1 Introduction
C.2 Provably Difficult Computational Problems Are not Suitable
C.3 One-way Functions and the Complexity Class UP

. . . complexity turns out to be most elusive precisely where it would be most welcome.
— C. H. Papadimitriou [229]

Real knowledge is to know the extent of one's ignorance.
— Confucius

The complex develops out of the simple.
— Colin Wilson

C.1 Introduction

It is worthwhile to ask the question why public-key cryptography must be based on problems that are only *believed* to be difficult. Complexity theory suggests concrete examples of *provably intractable* problems. This appendix provides a brief conceptual explanation why these provably difficult problems cannot be used for building cryptographic protocols. We may consequently conclude that at present we cannot prove a public-key cryptosystem to be secure. That is bad news, but we have to live with it.

Here, we make no attempts to furnish definitions of formal complexity classes. The excellent books by Papadimitriou [229] and by Sipser [280] can be consulted for that purpose. Here is a list of the complexity classes that we require for our discussion. The relationships between these classes are depicted in Figure C.1. All the containments shown in this figure are conjectured to be proper. With an abuse of notations we identify functional problems with decision problems.

Table C.1 Some complexity classes

Class	Brief description
P	Languages accepted by deterministic polynomial-time Turing machines
NP	Languages accepted by non-deterministic polynomial-time Turing machines
coNP	Complements of languages in NP
UP	Languages accepted by unambiguous polynomial-time Turing machines
PSPACE	Languages accepted by polynomial-space Turing machines
EXPTIME	Languages accepted by deterministic exponential-time Turing machines
EXPSPACE	Languages accepted by exponential-space Turing machines

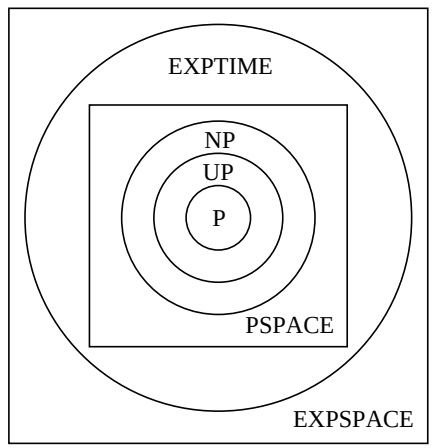


Figure C.1 Relations between complexity classes

C.2 Provably Difficult Computational Problems Are not Suitable

The $\mathrm{P} \stackrel{?}{=} \mathrm{NP}$ problem, arguably the deepest unsolved problem in theoretical computer science, may be suspected to have some bearing on public-key cryptography. Under the assumption that $\mathrm{P} \neq \mathrm{NP}$, one may feel tempted to go for using NP-complete problems for building secure cryptosystems. Unfortunately, this tempting invitation does not prove to be fruitful. Several cryptosystems based on NP-complete problems were broken and that is not really a surprise.

It may be the case that $\mathrm{P} = \mathrm{NP}$, and, if so, all NP-complete problems are solvable in polynomial time. It may, therefore, be advised to select problems that lie outside NP, that is, in strictly bigger complexity classes. By the time and space hierarchy theorems, we have $\mathrm{P} \subsetneq \mathrm{EXPTIME}$ and $\mathrm{PSPACE} \subsetneq \mathrm{EXPSPACE}$. Both EXPTIME and EXPSPACE have complete problems. An EXPTIME-complete problem cannot be solved in polynomial time, whereas an EXPSPACE-complete problem cannot be solved in polynomial space nor in polynomial time too. How about using these complete problems for designing cryptosystems? The idea may sound interesting, but these provably exponential problems turn out to be even poorer, perhaps irrelevant, for use in cryptography.

Let f_e and f_d be the encryption and decryption transforms for a public-key cryptosystem. We assume that the set of plaintext messages and the set of ciphertext messages are both finite. (Public-key cryptosystems are like block ciphers in this respect.) Moreover, since a ciphertext $c = f_e(m, e)$ is computable in polynomial time, the length of c is bounded by a polynomial in the length of m. An intruder can nondeterministically guess messages m (from the finite space) and check if $c = f_e(m, e)$ to validate the correctness of the guess. It, therefore, follows that deciphering a ciphertext message (with no additional information) is a problem in NP. That is the reason why we should not look beyond NP.

However, the full class NP, in particular, the most difficult (that is, complete) problems of NP, may be irrelevant for cryptography, as we argue in the next section. In other words, for building cryptosystems we expect to effectively exploit problems that are believed to be easier than NP-complete. Both the integer factoring and the discrete log problems are in the class $\mathrm{NP} \cap \mathrm{coNP}$. We have $\mathrm{P} \subseteq \mathrm{NP} \cap \mathrm{coNP}$. It is widely believed that this containment is proper. Also $\mathrm{NP} \cap \mathrm{coNP}$ is not known (nor expected) to have complete problems. Even if $\mathrm{P} \subsetneq \mathrm{NP} \cap \mathrm{coNP}$, both the factoring and the discrete log problems need not be outside P, since we are unlikely to produce completeness proofs for them. Only historical evidences exist, in favor of the fact that these two problems are difficult. The situation may change tomorrow. Complexity theory does not offer any formal protection.

EXERCISE SET C.2

C.1 Prove that the primality testing problem

$$\mathrm{PRIME} := \{p \in \mathbb{N} \mid p \text{ is prime}\}$$

is in $\mathrm{NP} \cap \mathrm{coNP}$.

(**Remark:** The AKS algorithm is a deterministic poly-time primality testing algorithm and therefore PRIME is in P and so trivially in NP ∩ coNP too. It can, however, be independently proved that primes have succinct certificates.)

C.2 Consider the decision version of the integer factorization problem:

$$\mathrm{DIFP} := \{(n,k) \in \mathbb{N}^2 \mid n \text{ has a factor } d \text{ in the range } 2 \leqslant d \leqslant k\}.$$

(a) Prove that DIFP ∈ NP ∩ coNP.
(b) Given a poly-time algorithm for DIFP, design a poly-time algorithm that factors an integer (that is, that solves the functional problem IFP).

C.3 Let G be a finite cyclic multiplicative group with a generator g. Assume that one can compute products in G in polynomial time. Consider the decision version of the discrete log problem in G:

$$\mathrm{DDLP} := \{(a,k) \in G \times \mathbb{N} \mid \operatorname{ind}_g a \leqslant k\}.$$

Here, indices ($\operatorname{ind}_g a$) are assumed to lie between 0 and $(\#G) - 1$.
(a) Prove that DDLP ∈ NP ∩ coNP.
(b) Given a poly-time algorithm for DDLP, design a poly-time algorithm that computes indices in G (that is, that solves the functional problem DLP in G).

C.3 One-way Functions and the Complexity Class UP

Any public-key encryption behaves like a one-way function, easy to compute but difficult to invert.

Definition C.1 Let Σ be an alphabet (a finite set of symbols). One may assume, without loss of generality, that $\Sigma = \{0,1\}$. Let Σ^* denote the set of all strings over Σ. A function $f : \Sigma^* \to \Sigma^*$ is called a *one-way function*, if it satisfies the following properties.

(1) f must be injective, that is, for every β the inverse $f^{-1}(\beta)$, if existent, is unique.
(2) For some real constant $k > 0$, we have $|\alpha|^{1/k} \leqslant |f(\alpha)| \leqslant |\alpha|^k$ for all $\alpha \in \Sigma^*$. (Here, $|\alpha|$ denotes the length of a string $\alpha \in \Sigma^*$.)
(3) f can be computed in deterministic polynomial time, that is, $f \in \mathrm{P}$.
(4) f^{-1} must not be computable in polynomial time[1], that is, $f^{-1} \notin \mathrm{P}$. In view of Property (2), we have $f^{-1} \in \mathrm{NP}$. So we require $f^{-1} \in \mathrm{NP} \setminus \mathrm{P}$. □

Property (1) ensures unique decryption. Property (2) implies that the length of $f(\alpha)$ is polynomially bounded both above and below by the length of α. Property (3) suggests ease of encryption, whereas Property (4) suggests difficulty of decryption.

We do not know whether there exists a one-way function. The following functions are strongly suspected to be one-way. However, we do not seem to have any clues about how we can prove these functions to be one-way.

[1] A stronger (but essential) requirement is that f^{-1} must not be computable by polynomial-time probabilistic algorithms.

Example C.1

(1) The function that multiplies two primes p, q with $p < q$ is believed to be one-way. Computing its inverse is the RSA integer factoring problem.

(2) The discrete exponentiation function in a finite field $\mathbb{F}_q$, that maps $x \in \mathbb{N}$, $0 \leqslant x \leqslant q-2$, to $g^x \in \mathbb{F}_q^*$ for some fixed $g \in \mathbb{F}_q^*$ is suspected to be one-way. Its inverse is the discrete logarithm function.

(3) The RSA encryption function $m \mapsto m^e \pmod{n}$ for some fixed parameters n, e is alleged to be one-way. Its inverse is RSA decryption. □

It is evident that if $\mathrm{P} = \mathrm{NP}$, there cannot exist one-way functions. The converse of this is not true, that is, even if $\mathrm{P} \neq \mathrm{NP}$, there may exist no one-way functions.

Definition C.2 A non-deterministic Turing machine which has at most one accepting branch of computation for every input string is called an *unambiguous Turing machine*. The class of languages accepted by poly-time unambiguous Turing machines is denoted by UP. □

Clearly, $\mathrm{P} \subseteq \mathrm{UP} \subseteq \mathrm{NP}$. Both the containments are assumed to be proper. The importance of the class UP stems from the following result:

Theorem C.1 There exists a one-way function if and only if $\mathrm{P} \neq \mathrm{UP}$. ■

Therefore, the $\mathrm{P} \stackrel{?}{=} \mathrm{UP}$ question is relevant for cryptography and not the $\mathrm{P} \stackrel{?}{=} \mathrm{NP}$ question. The class UP is not known (nor expected) to have complete problems. So locating a one-way function may be a difficult task. But at the minimum we are now in the right track.[2] Complexity theory helped us shift our attention from NP (or bigger classes) to UP.

In order to use a one-way function f for cryptographic purposes, we require additional properties of f. Computing f^{-1} must be difficult for an intruder, whereas the same computation ought to be easy to the legitimate recipient. Thus, f must support poly-time inversion, provided that some secret piece of information (the trapdoor) is available during the computation of the inverse. A one-way function with a trapdoor is called a *trapdoor one-way function*.

The first two functions of Example C.1 do not have obvious trapdoors and so cannot be straightaway used for designing cryptosystems. The third function (RSA encryption) has the requisite trapdoor, namely, the decryption exponent d satisfying $ed \equiv 1 \pmod{\phi(n)}$.

The hunt for a theoretical foundation does not end here. It begins. Most part of complexity theory deals with *worst-case* complexities of problems, rather than their *average* or *expected* complexities. A one-way function, even if existent, may be difficult to invert for only few instances, whereas cryptography demands the inversion problem to be difficult for most instances. A function meeting even this cryptographic demand need not be suitable, since there may be reductions to map hard instances to easy instances. Moreover, the trapdoors themselves may inject vulnerabilities and prepare room for quick attacks.

There still remains a long way to go!

[2] Well, hopefully!

EXERCISE SET C.3

C.4 Let $f : \Sigma^* \to \Sigma^*$ be a function with the property that $f(f(\alpha)) = f(\alpha)$ for every $\alpha \in \Sigma^*$. Argue that f is *not* a one-way function.

C.5 Design unambiguous polynomial time Turing machines for computing the inverses of the functions described in Example C.1.

C.6 Show that if there exists a *bijective* one-way function, then $\mathrm{NP} \cap \mathrm{coNP} \neq \mathrm{P}$. **[H]**

Hints to Selected Exercises

D

The greatest thing in family life is to take a hint when a hint is intended and not to take a hint when a hint isn't intended.
— Robert Frost

Teachers open the door, but you must enter by yourself.
— Chinese Proverb

Imagination grows by exercise, and contrary to common belief, is more powerful in the mature than in the young.
— W. Somerset Maugham

2.11 (a) Apply Theorem 2.3 to the restriction to H of the canonical homomorphism $G \to G/K$.

2.11 (b) Apply Theorem 2.3 to the canonical homomorphism $G/H \to G/K$, $aH \mapsto aK$, $a \in G$.

2.14 (c) Consider the canonical surjection $G \to G/H$.

2.17 (a) Let $i \neq j$ and $g \in H_i \cap H_j$. Then $\operatorname{ord} g$ divides both $p_i^{\alpha_i}$ and $p_j^{\alpha_j}$ and so is equal to 1, that is, $g = e$. Now let $h_i, h_i' \in H_i$ and $h_j, h_j' \in H_j$ with $h_i h_j = h_i' h_j'$. But then $h_i^{-1} h_i' = h_j (h_j')^{-1} \in H_i \cap H_j = \{e\}$. Thus $\#(H_i H_j) = (\#H_i)(\#H_j)$. Generalize this argument to show that $\#(H_1 \cdots H_r) = n$.

2.18 First consider the special case $\#G = p^r$ for some $p \in \mathbb{P}$ and $r \in \mathbb{N}$. For each $g \in G$, the order $\operatorname{ord}_G g$ is of the form p^{s_g} for some $s_g \leqslant r$. Let s be the maximum of the values s_g, $g \in G$. Take any element $h \in G$ with $\operatorname{ord}_G h = p^s$. Then $e, h, \ldots, h^{p^s - 1}$ are *all* the elements x that satisfy $x^{p^s} = e$. But by the choice of s every element $x \in G$ satisfies $x^{p^s} = e$. Hence we must have $s = r$. This proves the assertion for the special case. For the general case, use this special case in conjunction with Exercise 2.17.

2.19 (b) Show that $\varphi : H_1 \times \cdots \times H_r \to G$, $(h_1, \ldots, h_r) \mapsto h_1 \cdots h_r$, is a group isomorphism.

2.23 Use Zorn's lemma.

2.24 (c) Let $\mathfrak{N}_R$ be the intersection of all prime ideals of R. First show that $\mathfrak{n}_R \subseteq \mathfrak{N}_R$. To prove the reverse inclusion take $f \notin \mathfrak{n}_R$ and consider the set S of all non-unit ideals $\mathfrak{a}$ of R such that $f^n \notin \mathfrak{a}$ for all $n \in \mathbb{N}$. If f is a non-unit, the set S is non-empty and by Zorn's lemma has a maximal element, say $\mathfrak{p}$. Show that $\mathfrak{p}$ is a prime ideal of R.

2.25 For $0 \neq a \in R$, the map $R \to R$, $b \mapsto ab$, is injective and hence surjective by Exercise 2.4.

2.30 Apply the isomorphism theorem to the canonical surjection $R/\mathfrak{a} \to R/\mathfrak{b}$, $a + \mathfrak{a} \mapsto a + \mathfrak{b}$.

2.33 [(1)⇒(2)] Let $\mathfrak{a}_1 \subseteq \mathfrak{a}_2 \subseteq \cdots$ be an ascending chain of ideals of R. Consider the ideal $\bigcup_{n \in \mathbb{N}} \mathfrak{a}_n$ which is finitely generated by hypothesis.
[(3)⇒(1)] Let $\mathfrak{a}$ be an ideal of R. Consider the set of all finitely generated ideals of R contained in $\mathfrak{a}$.

2.36 Use the *pigeon-hole principle*: If there are $n + 1$ pigeons in n holes, then there exists at least one hole containing more than one pigeons.

2.37 Consider the integer $t \in \mathbb{N}$ satisfying $2^t \leqslant n < 2^{t+1}$.

2.39 (e) $1^2 \equiv (n-1)^2 \pmod{n}$.

2.39 (f) Apply Wilson's theorem.

2.40 Use Fermat's little theorem.

2.41 Use Wilson's theorem or Euler's criterion.

2.45 Reduce to the case $y^2 \equiv \alpha \pmod{p}$.

2.49 (a) Consider the canonical group homomorphism $\mathbb{Z}_{2^e}^* \to \mathbb{Z}_8^*$ and the fact that a surjective group homomorphism from a cyclic group G onto G' implies that G' is cyclic.

2.49 (b) Let $a \in \mathbb{Z}$ be a primitive element modulo p. The residue class of a in $\mathbb{Z}_{p^e}^*$ has order $k(p-1)$ for some $k \in \mathbb{N}$. Show that the order of $b := p + 1$ modulo p^e is p^{e-1}. So the order of $a^k b$ modulo p^e is $p^{e-1}(p-1) = \phi(p^e)$.

2.50 Use the Chinese remainder theorem in conjunction with Exercises 2.20 and 2.49.

2.53 Take $f_i(X) := \prod_{j \neq i} (X - a_j)/(a_i - a_j)$. The interpolating polynomial is $f(X) := \sum_{i=0}^{n} b_i f_i(X)$. Use Exercise 2.52 to establish the uniqueness.

2.56 (b) $f(X) \in \mathbb{Z}[X]$ is irreducible in $\mathbb{Z}[X]$ if and only if $f(X+1)$ is irreducible in $\mathbb{Z}[X]$.

2.58 Use the fundamental theorem of algebra.

2.63 Consider the set $\mathcal{L}$ of all linearly independent subsets of V that contain T. Show that every chain in $\mathcal{L}$ has an upper bound in $\mathcal{L}$. By Zorn's Lemma, there exists a maximal element $S \in \mathcal{L}$. Show that S generates V.

2.64 (b) Use Exercise 2.63.

2.68 Let $p_1, \ldots, p_n$ be n distinct primes. Take $a := \prod_{i=1}^{n} p_i$ and $a_i := a/p_i$ for $i = 1, \ldots, n$.

2.72 (a) If N is the $\mathbb{Z}$-submodule of $\mathbb{Q}$ generated by a_i/b_i, $i = 1, \ldots, n$, with $\gcd(a_i, b_i) = 1$, then for any prime p that does not divide $b_1 \cdots b_n$ we have $1/p \notin N$.

2.72 (b) Any two distinct elements of $\mathbb{Q}$ are linearly dependent over $\mathbb{Z}$. Now use Exercise 2.69.

2.74 (b) Let the conjugates of $\alpha \in K$ over F be $\alpha_1 = \alpha, \alpha_2, \ldots, \alpha_n$. Since φ is injective, it follows from (a) that φ makes a permutation of $\alpha_1, \ldots, \alpha_n$. So φ is surjective.

2.75 (a) Use Exercise 2.61.

2.76 (b) The *if* part follows from Exercise 2.61. For proving the *only if* part, take $a \in F$. If the polynomial $f(X) := X^p - a$ splits over F, we are done. So suppose that there exists an irreducible divisor $g(X) \in F[X]$ of $f(X)$ of degree $\geqslant 2$. By the separability of F, there exist two distinct roots α, β of $g(X)$. Let $K := F(\alpha, \beta)$. Show that the *Frobenius map* $\mathfrak{F} : K \to K$, $\gamma \mapsto \gamma^p$, is an endomorphism of K. Also there exists a field isomorphism $\tau : F(\alpha) \to F(\beta)$ which fixes F element-wise and takes $\alpha \mapsto \beta$. But then $\mathfrak{F}(\beta) = \beta^p = \tau(\alpha)^p = \tau(\alpha^p) = \tau(a) = a = \alpha^p = \mathfrak{F}(\alpha)$. Since any field homomorphism is injective, α equals β, a contradiction. Thus no $g(X)$ chosen as above can exist.

2.77 (a) Let $g(X) \in F[X]$ be an irreducible polynomial with $g(\alpha) = 0$ for some $\alpha \in K$. Let β be another root of g. We show that $\beta \in K$. By Lemma 2.5, there is an isomorphism $\mu : F(\alpha) \to F(\beta)$. Clearly, K is the splitting field of f over $F(\alpha)$. Let K' be the splitting field of $\mu^*(f)$ over $F(\beta)$. By Proposition 2.33, $K \cong K'$. If $\gamma_1, \ldots, \gamma_d \in K$ are the roots of f, then $K' \cong F(\beta, \gamma_1, \ldots, \gamma_d) = K(\beta)$. But then $K \cong K(\beta)$.

2.78 (a) Consider transcendental numbers.

2.78 (b) Let $\varphi \in \mathrm{Aut}_{\mathbb{Q}} \mathbb{R}$. For $a \in \mathbb{R}_+$, we have $\varphi(a) = \varphi(\sqrt{a})^2 \geqslant 0$, implying that $\varphi(a) \leqslant \varphi(b)$ for $a, b \in \mathbb{R}$ with $a \leqslant b$. Now assume $a < \varphi(a)$ for some $a \in \mathbb{R} \setminus \mathbb{Q}$. Choose a rational number b with $a < b < \varphi(a)$. Then $\varphi(a) \leqslant \varphi(b) = b$, a contradiction. Thus $\varphi(a) \geqslant a$. Similarly $\varphi(a) \leqslant a$.

2.80 Use the binomial theorem and induction on n.

2.82 Follow the proof of Theorem 2.37.

2.90 Example 2.18.

2.91 (b) By the fundamental theorem of Galois theory, $\#\,\mathrm{Aut}_{\mathbb{F}_q} \mathbb{F}_{q^m} = [\mathbb{F}_{q^m} : \mathbb{F}_q] = m$. Now show that $\mathrm{id}_{\mathbb{F}_{q^m}}, \mathfrak{F}_q, \ldots, \mathfrak{F}_q^{m-1}$ are distinct $\mathbb{F}_q$-automorphisms of $\mathbb{F}_{q^m}$.

2.92 (a) Assume $r > 1$. We have the extensions $\mathbb{F}_q \subseteq \mathbb{F}_{q^r} \subseteq \mathbb{F}_{q^d}$, where $\mathbb{F}_{q^d}$ is the splitting field of f over $\mathbb{F}_q$ and hence over $\mathbb{F}_{q^r}$. Consider the minimal polynomial of a root $\alpha \in \mathbb{F}_{q^d}$ of f over $\mathbb{F}_{q^r}$. Conversely, let f be reducible over $\mathbb{F}_{q^m}$. Choose an irreducible factor $h \in \mathbb{F}_{q^m}[X]$ of f with $\deg h = s < d$. Now h has one (and hence all) roots in $\mathbb{F}_{q^{sm}}$ and, therefore, $d | sm$.

2.93 Use Corollary 2.18.

2.98 In each case, the defining polynomial is quadratic in Y (and with coefficients in $K[X]$). If this polynomial admits a non-trivial factorization, one can reach a contradiction by considering the degrees of X in the coefficients of Y^1 and Y^0.

2.103 For simplicity, consider the case $\mathrm{char}\, K \neq 2, 3$. Show that the curves $Y^2 + Y = X^3$ and $Y^2 = X^3 + X$ have j-invariants 0 and 1728 respectively. Finally, if $\bar{\jmath} \neq 0, 1728$, then the curve $Y^2 + XY = X^3 - (36/(\bar{\jmath} - 1728))X - (1/(\bar{\jmath} - 1728))$ has j-invariant $\bar{\jmath}$. One must also argue that these are actually elliptic curves, that is, have non-zero discriminants.

2.111 Use Theorem 2.51.

2.112 (a) Pair a point with its opposite. This pairing fails for points of orders 1 and 2.

2.112 (c) Consider the elliptic curve $E : Y^2 = X^3 + 3$ over $\mathbb{F}_{13}$. We have $\#E(\mathbb{F}_{13}) \cong \mathbb{Z}_3 \oplus \mathbb{Z}_3$, whereas $X^3 + 3$ is irreducible modulo 13.

2.113 (a) Every element of $\mathbb{F}_{2^m}$ has a unique square root.
2.115 (a) Use Theorem 2.49 or Exercise 2.17.
2.115 (b) Use Theorem 2.50.
2.115 (c) The trace of Frobenius at q is 0 in this case. Now, use Theorem 2.50.
2.123 Factor $\mathrm{N}(G)$ in $\bar{K}[x]$.
2.127 Let $\mathfrak{a} = Ax_1 + \cdots + Ax_n$, $x_i \in \mathfrak{a}$. For each i, write $\gamma x_i = \sum_{j=1}^{n} a_{ij}x_j$, $a_{ij} \in A$. But then $\det \mathfrak{A} = 0$, where $\mathfrak{A} = (\gamma\delta_{ij} - a_{ij})_{1\leqslant i,j\leqslant n}$, δ_{ij} being the Kronecker delta.
2.128 (b) Use Part (a) and Exercise 2.126(c).
2.128 (c) Let $S := A \setminus \mathfrak{p}$. By Exercise 2.130, $B_{\mathfrak{p}} := S^{-1}B$ is integral over $A_{\mathfrak{p}} = S^{-1}A$. Let $\mathfrak{m}$ be the ideal generated by $\mathfrak{p}$ in $A_{\mathfrak{p}}$ and let $\mathfrak{n}$ and $\mathfrak{n}'$ be the ideals of $B_{\mathfrak{p}}$ generated respectively by $\mathfrak{q}$ and $\mathfrak{q}'$. Now, use Part (b).
2.133 (b) In a PID, non-zero prime ideals are maximal.
2.137 (a) Since $\mathfrak{q}_1$ and $\mathfrak{q}_2$ are maximal, we have $\mathfrak{q}_1 + \mathfrak{q}_2 = A$, that is, $a_1 + a_2 = 1$ for some $a_1 \in \mathfrak{q}_1$ and $a_2 \in \mathfrak{q}_2$. Now use the fact that $(a_1 + a_2)^{e_1+e_2} = 1$.
2.137 (b) Use CRT.
2.138 (a) Since $\mathfrak{a}$ is invertible, $\mathfrak{a}\mathfrak{b}' = A$ for some fractional ideal $\mathfrak{b}'$.
2.140 (a) For $\mathfrak{a} \neq 0$, let $\alpha_1, \ldots, \alpha_{\mathrm{N}(\mathfrak{a})}$ constitute a complete residue system of $\mathfrak{O}$ modulo $\mathfrak{a}$. Then $1 + \alpha_1, \ldots, 1 + \alpha_{\mathrm{N}(\mathfrak{a})}$ also form a complete residue system of $\mathfrak{O}$ modulo $\mathfrak{a}$.
2.142 (d) Take $\alpha := \frac{1+\sqrt{-6}}{2}$ in Part (b).
2.143 (a) Reduce modulo 4.
2.143 (c) Let $a + b\sqrt{-2} \in \mathfrak{O}_K$ divide this gcd. Then $a + b\sqrt{-2}$ divides $2y$ and $2\sqrt{-2}$. Take norms.
2.144 (b) Look at the expansion of $a - 1$ in base p. More precisely, let $a < p^N$ for some $N \in \mathbb{N}$. Then $-a = (p^N - a) - p^N = [(p^N - 1) - (a - 1)] - p^N$.
2.152 (c) First show that $\sum_{i=1}^{n} i \cdot i! = (n + 1)! - 1$.
2.153 Use unique factorization of rationals.
2.154 Show by induction on n that p^{n+1} divides $a^{p^{n+1}} - a^{p^n}$ in $\hat{\mathbb{Z}}_p$ for all $n \in \mathbb{N}$.
2.161 There exists an irreducible polynomial in $\mathbb{F}_p[X]$ of every degree $d \in \mathbb{N}$.
3.7 The implication DLP $\preceq$ GDLP is obvious. For the reverse implication, use Proposition 2.5.
3.18 (b) Consider the binary expansion of m.
3.19 if n is a pseudoprime to base a and not a pseudoprime to base b, then n is not a pseudoprime to base ab.
3.20 (a) If $p^2|n$ for some $p \in \mathbb{P}$, take $a \in \mathbb{Z}_n^*$ with $\mathrm{ord}_n(a) = p$. If n is square-free, consider a prime divisor p of n and take $a \in \mathbb{Z}_n^*$ with $\left(\frac{a}{p}\right) = -1$ and $a \equiv 1 \pmod{n/p}$.
3.20 (b) if n is an Euler pseudoprime to base a and not an Euler pseudoprime to base b, then n is not an Euler pseudoprime to base ab.
3.21 (a) Let $n = p_1^{\alpha_1} \cdots p_r^{\alpha_r}$ be the prime factorization of n with r and each α_i in $\mathbb{N}$. Then, $\mathbb{Z}_n^* = \prod_{i=1}^{r} \mathbb{Z}_{p_i^{\alpha_i}}^*$. For odd p_i, the group $\mathbb{Z}_{p_i^{\alpha_i}}^*$ is cyclic of order $p_i^{\alpha_i-1}(p_i - 1)$ and hence contains an element of order $p_i - 1$.
3.21 (b) $\mathrm{ord}_n(-1) = 2$.
3.21 (c) Let $\mathrm{v}_p(n) \geqslant 2$ for some odd prime p. Construct an element $a \in \mathbb{Z}_n^*$ with $\mathrm{ord}_n(a) = p$.
3.28 Proceed by induction on $i = 1, \ldots, r$. For $1 \leqslant i \leqslant r$, define $\nu_i := n_1 \cdots n_i$ and let $b_i \in \mathbb{Z}_{\nu_i}$ be a solution of the congruences $b_i \equiv a_j \pmod{n_j}$ for $j = 1, \ldots, i$. If $i < r$, use the combining formula given in Section 2.5 to find $b_{i+1} \in \mathbb{Z}_{\nu_{i+1}}$ such that $b_{i+1} \equiv b_i \pmod{\nu_i}$ and $b_{i+1} \equiv a_{i+1} \pmod{n_{i+1}}$.
3.31 Apply Newton's iteration to compute a zero of $x^2 - n$.

3.32 (a) Apply Newton's iteration to compute a zero of $x^k - n$.

3.34 (b) The updating $d(X) := d(X) - X^{i-s}b(X)$ needs to consider only the non-zero words of b.

3.36 (b) First consider $b = 0$ and note that the roots of $X^{(q-1)/2} - 1$ (resp. $X^{(q-1)/2} + 1$) are all the quadratic residues (resp. non-residues) of $\mathbb{F}_q$.

3.36 (c) First consider $b = 0$.

3.40 For $a \in G$, we have $\operatorname{ord}(a)|m$ and for each $i = 1, \ldots, r$ the multiplicity $\mathrm{v}_{p_i}(\operatorname{ord}(a))$ is the smallest of the non-negative integers k satisfying $a^{n/p_i^{\alpha_i - k}} = e$.

3.41 (a) Use the CRT.

3.43 (a) Use the CRT and the fact that $\left(\frac{-1}{r}\right) = -1$ for an odd prime $r \equiv 3 \pmod 4$.

4.1 (a) Using the CRT, reduce to the case that n is prime. Then φ is bijective $\iff$ the restriction $\varphi^* := \varphi|\mathbb{Z}_n^* : \mathbb{Z}_n^* \to \mathbb{Z}_n^*$ is bijective. Now, if $\gcd(a, \phi(n)) = 1$, the inverse of φ^* is given by $\psi^* : x \mapsto x^b$, where $ab \equiv 1 \pmod{\phi(n)}$. On the other hand, if q is a prime divisor of $\gcd(a, \phi(n))$, choose an element $y \in \mathbb{Z}_n^*$ with $\operatorname{ord}(y) = q$. But then $y^a \equiv 1 \pmod n$, that is, φ^* is not injective. This exercise provides the foundation for the RSA cryptosystems.

4.1 (b) In view of the CRT, reduce to the case $n = p^\alpha$ for $p \in \mathbb{P}$ and $\alpha > 1$. Then $(p^{\alpha-1})^a \equiv 0 \pmod n$.

4.6 Consider the integral $\int_{x=1}^{m} \frac{\mathrm{d}x}{x} = \ln m$.

4.9 Use the CRT and lifting.

4.10 For proving IFP $\preceq$ SQRTP, let n be an odd composite integer, choose a random $y \in \mathbb{Z}_n^*$ and compute a square root x of y^2 modulo n. By Exercise 4.9, the probability that $x \equiv \pm y \pmod n$ is at most $1/2$.

4.12 (d) Eliminate a from $T(a, b, c)$ using $a + b + c = 0$. For each *fixed* c, allow b to vary and use a sieve to find out all the values of b for which $T(a, b, c)$ is smooth for the fixed c.

4.13 You may use the prime number theorem and the fact that the sum $\frac{1}{q_1} + \frac{1}{q_2} + \cdots + \frac{1}{q_t}$ of the reciprocals of the first t primes asymptotically approaches $\ln \ln t$.

4.15 If $a < a_1$ or $a > a_m$, then no i exists. So assume that $a_1 \leqslant a \leqslant a_m$ and let $d := \lfloor (1 + m)/2 \rfloor$. If $a = a_d$, return d, else if $a < a_d$, recursively search a among the elements $a_1, \ldots, a_{d-1}$, and if $a > a_d$, recursively search a among the elements $a_{d+1}, \ldots, a_m$.

4.16 (a) Use Lagrange's interpolation formula (Exercise 2.53).

4.18 (a) One may precompute the values $\sigma_i := p \operatorname{rem} q_i$, $i = 1, \ldots, t$. Note that $q_i|(g^\alpha + kp)$ if and only if $\rho_{k,i} = 0$.

4.19 (a) Use the approximation $T(c_1, c_2) \approx (c_1 + c_2)H$.

4.21 (c) $T(a, b, c) = -b^2 - c(x + cy)b + (z - c^2 x)$.

4.21 (d) Imitate the second stage of the LSM.

4.23 Let the factor base consist of all irreducible polynomials over $\mathbb{F}_2$ of degrees $\leqslant m$ together with the polynomials of the form $X^k + h(X)$, $h \in \mathbb{F}_2[X]$, $\deg h \leqslant m$. The optimal running time $L[\sqrt{2/3}] \approx L[0.816]$ of this algorithm corresponds to $m \approx \sqrt{\frac{n \ln n}{6 \ln 2}}$.

4.24 (b) $X^m - 1 \in K[X]$ is square-free.

4.24 (c) Use the fact $X^m - 1 = (X^{m/p^{\mathrm{v}_p(m)}} - 1)^{p^{\mathrm{v}_p(m)}}$.

4.24 (d) Theorem 2.39.

4.25 (a) Look at the roots of the polynomials on the two sides.

4.25 (c) If $\operatorname{ord} \omega = m$, then $\operatorname{ord}(-\omega) = 2m$.

4.25 (d) $\omega, \omega^q, \ldots, \omega^{q^{l-1}}$ are all the roots of the minimal polynomial of ω over $\mathbb{F}_q$.

4.26 (b) Use the Mordell–Weil theorem.

4.26 (c) Use Theorem 4.2.

5.2 (a) Solve the simultaneous congruences $x \equiv c_i \pmod{n_i}$, $i = 1, \ldots, e$, and then take the *integer* e-th root of the solution x, $1 \leqslant x \leqslant n_1 \cdots n_e$.

5.2 (b) Append (different) pseudorandom bit strings to m before encryption. This process is often referred to as *salting*.

5.3 (a) In view of the Chinese remainder theorem, reduce to the case $n = p^r$ for some $p \in \mathbb{P}$ and $r \in \mathbb{N}$.

5.4 $ue_1 + ve_2 = 1$ for some $u, v \in \mathbb{Z}$.

5.6 If the same session key is used to generate the ciphertext pairs (r_1, s_1) and (r_2, s_2) on two plaintext messages m_1 and m_2, then $m_1/m_2 = s_1/s_2$.

5.7 (c) Let $x = (x_{l-1} \ldots x_1 x_0)_2$. Define $x' := (x_{l-1} \ldots x_2 x_1)_2$ and $y' := g^{x'} \pmod{p}$. Then, $y \equiv y'^2 g^{x_0} \pmod{p}$. Since x_0 is easily computable, y' can be obtained by obtaining a square root of y modulo p. Argue that a call of the oracle helps us choose the correct square root y' of y. Now, use recursion.

5.8 Let g' be any randomly chosen generator of $\mathbb{F}_q^*$, where $q := p^h$. One computes $a'_i := \operatorname{ind}_{g'}(x + i)$ for $i = 0, 1, \ldots, p-1$. We then have the equality of the sets

$$\{a'_0, a'_1, \ldots, a'_{p-1}\} = \{l(\alpha_0 - d), l(\alpha_1 - d), \ldots, l(\alpha_{p-1} - d)\}$$

modulo $q - 1$, where $l := \operatorname{ind}_{g'} g$. But then for each i we have a (yet unknown) j such that $a'_j = l(\alpha_i - d)$. Show that trying all possibilities for i and j one can effectively recover l and hence $g = g'^l$ and hence π.

5.9 Let g', a'_i and l be as in Exercise 5.8. Now, we have the equality of the sets

$$\{a'_0 - a'_0, a'_1 - a'_0, \ldots, a'_{p-1} - a'_0\} = \{l(\alpha_0 - \alpha_0), l(\alpha_1 - \alpha_0), \ldots, l(\alpha_{p-1} - \alpha_0)\}$$

modulo $q - 1$.

5.11 $u_i - u_1 \equiv \alpha^{-1}(c_i - c_1) \circledast h^{-1} \pmod{\beta}$ are polynomials with small coefficients.

5.15 (a) If Alice generates the signatures (M_1, s_1) and (M_2, s_2) on two messages M_1 and M_2, then her signature on a message M with $H(M) \equiv H(M_1)H(M_2) \pmod{n}$ is $s_1 s_2 \pmod{n}$. Thus, without knowing the private key of Alice, an intruder can generate a valid signature $(M, s_1 s_2)$ of Alice, provided that such an M can be computed. Of course, here the intruder has little control over the message M. The PKC standards form RSA Laboratories add some redundancy to the hash function output before signing. The product of two hash values with redundancy is, in general, expected not to have the redundancy. This increases the security of the scheme against existential forgeries beyond that provided by the first pre-image resistance of the underlying hash function.

5.15 (b) For any $s \in \mathbb{Z}_n$, a valid signature is (M, s), where $H(M) \equiv s^2 \pmod{n}$.

5.15 (c) Choose random integers u, v with $\gcd(v, n) = 1$ and take $d' := u + dv$. Of course, d and hence d' are unknown to Carol, but she can compute $s = g^{d'} = g^u (g^d)^v$ and $t \equiv -H(s)v^{-1} \pmod{n}$. But then (M, s, t) is a valid ElGamal signature on a message M for which $H(M) \equiv tu \pmod{n}$.

5.16 Obviously, c itself could be a possible choice, but that is not random and Bob might refuse to sign c. Carol should hide c by $cr^e \pmod{n}$ for some randomly chosen r known to her.

5.23 (a) $\mathbb{Z}_n^* \cong \mathbb{Z}_p^* \times \mathbb{Z}_q^*$ by the CRT.

5.25 (a) Replace the random challenge of the verifier by the hash value of the string obtained by concatenating the message to be signed with the witness.

5.26 (d) Bob finds a random b' with $\left(\frac{b'}{n}\right) = -1$ and sends $a := (b')^2 \pmod{n}$ to Alice. But then Alice's response b yields a non-trivial factor $\gcd(b - b', n)$ of n.

7.5 $s \equiv \tilde{s}s_j^{2d_j-1} \pmod{n}$ and $m \equiv s^e \pmod{n}$.

7.9 (a) Use Exercise 2.44(b).

7.9 (c) Again use Exercise 2.44(b).

7.9 (d) Use Part (c) in conjunction with the CRT, and separately consider the three cases $\mathrm{v}_2(p-1) = \mathrm{v}_2(q-1)$, $\mathrm{v}_2(p-1) > \mathrm{v}_2(q-1)$ and $\mathrm{v}_2(p-1) < \mathrm{v}_2(q-1)$.

A.2 $e(\overline{X}, \overline{J}) = e(X, J)$ for all X, J. One does not have to look at the S-boxes for proving this.

A.9 (c) For $i = 0, 1, 2, 3, 4N_r, 4N_r + 1, 4N_r + 2, 4N_r + 3$, take $K'_i := K_i$. For other values of i, take $K'_i := \mathrm{MixCols}^{-1}(K_i)$.

A.14 (b) Let $D_L(X) := X^d C_L(1/X) = a_0 + a_1X + a_2X^2 + \cdots + a_{d-1}X^{d-1} + X^d$. Consider the $\mathbb{F}_2$-algebra $A := \mathbb{F}_2[X]/\langle D_L(X)\rangle = \mathbb{F}_2[x] \cong (\mathbb{F}_2)^d$, where $x := X + \langle D_L(X)\rangle$. The $\mathbb{F}_2$-linear transformation $\lambda_x : A \to A$ defined by $g(x) \mapsto xg(x)$ has the matrix Δ_L with respect to the polynomial basis $(1, x, \ldots, x^{d-1})$. If $f(X) \in \mathbb{F}_2[X]$ is the minimal polynomial of λ_x, then $[f(\lambda_x)](1) = f(x) = 0$. Now, use the fact that $1, x, \ldots, x^{d-1}$ are linearly independent over $\mathbb{F}_2$.

A.16 (b) [only if] Take $\sigma \neq 00\ldots01$. Since σ is non-zero, $s_i = 1$ for some $i \in \{0, 1, \ldots, d-2\}$. Construct an LFSR with $d-1$ stages initialized to $s_0 s_1 \ldots s_{d-2}$ to generate σ.

A.19 Suppose that we want to compute a second pre-image for $H_2(x)$. If $x \in S$, any $x' \in S \setminus \{x\}$ is a second pre-image for $H_2(x)$. If $x \notin S$, computing a second pre-image for $H_2(x)$ is equivalent to computing a second pre-image for $H(x)$. The density of the (finite) set S is 0 in the (infinite) set of all bit strings. Thus, H_2 is second pre-image resistant. On the other hand, for any two distinct $x, x' \in S$ we have a collision (x, x') for H_2.

A.20 Collision resistance of H implies that of H_3. On the other hand, for a positive fraction (half) of the $(n+1)$-bit strings y, it is easy to compute a pre-image of y under H_3.

A.21 If y is a square root of a modulo m, then so is $m - y$ too.

A.22 Use the birthday paradox (Exercise 2.172).

A.23 (d) Let $L := F_1(L')$ and $R := F_1(R')$ with both R and R' non-zero. Then, $F_1(L \,||\, R) = F_2(L' \,||\, R')$.

A.25 Let $\mathbf{h}^{(i)}$ denote the column vector of dimension 160 having the bits of $H^{(i)}$ as its elements and $\mathbf{m}^{(i)}$ the column vector of dimension $512 + 160 = 672$ having the bits of $M^{(i)}$ and of $H^{(i)}$ as its elements. Show that the modified design of SHA-1 leads to the relation $\mathbf{h}^{(i)} \equiv A\mathbf{m}^{(i-1)} + \mathbf{c} \pmod{2}$ for some constant 160×672 matrix A over $\mathbb{F}_2$ and for some constant vector $\mathbf{c}$. So what then?

C.6 For $\alpha, \beta \in \Sigma^*$, call $\alpha \leqslant \beta$ if and only if $|\alpha| < |\beta|$ or $|\alpha| = |\beta|$ and α is lexicographically smaller than β. This $\leqslant$ produces a well-ordering of Σ^*. For a one-way function f, look at the language $L_f := \{(\alpha, \beta) \in \Sigma^* \times \Sigma^* \mid f(\gamma) = \alpha \text{ for some } \gamma \in \Sigma^* \text{ with } \gamma \leqslant \beta\}$.

This page is intentionally left blank

References

If you steal from one author, it's plagiarism; if you steal from many, it's research.
— Wilson Mizner

Literature is the question minus the answer.
— Roland Barthes

Everything that can be invented, has been invented.
— Charles H. Duell, 1899

[1] Adkins, W. A. and S. H. Weintraub (1992). *Algebra: An Approach via Module Theory*. *Graduate Texts in Mathematics*, 136. New York: Springer.

[2] Adleman, L. M., J. DeMarrais and M.-D. A. Huang (1994). "A Subexponential Algorithm for Discrete Logarithms over the Rational Subgroup of the Jacobians of Large Genus Hyperelliptic Curves over Finite Fields", *Algorithmic Number Theory—ANTS-I*, *Lecture Notes in Computer Science*, 877. pp. 28–40. Berlin/Heidelberg: Springer.

[3] Adleman, L. M. and M.-D. A. Huang (1992). "Primality Testing and Two Dimensional Abelian Varieties over Finite Fields", *Lecture Notes in Mathematics*, 1512. Berlin: Springer.

[4] Adleman, L. M., C. Pomerance and R. S. Rumely (1983). "On Distinguishing Prime Numbers from Composite Numbers", *Annals of Mathematics*, 117: 173–206.

[5] Agarwal, M., N. Kayal and N. Saxena (2002), "Primes Is in P" [online document]. Available at http://www.cse.iitk.ac.in/users/manindra/algebra/primality_v6.pdf (October 2008).

[6] * Ahlfors, L. V. (1966). *Complex Analysis*. New York: McGraw-Hill.

[7] * Aho, A. V., J. E. Hopcroft and J. D. Ullman (1974). *The Designs and Analysis of Algorithms*. Reading, Massachusetts: Addison-Wesley.

[8] * ——— (1983). *Data Structues and Algorithms*. Reading, Massachusetts: Addison-Wesley.

[9] Aigner, M. and E. Oswald (2007), "Power Analysis Tutorial" [online document]. Available at http://www.iaik.tugraz.at/content/research/implementation_attacks/introduction_to_impa/dpa_tutorial.pdf (October 2008).

[10] Akkar, M.-L., R. Bevan, P. Dischamp and D. Moyart (2000). "Power Analysis, What Is Now Possible", *Advances in Cryptology—ASIACRYPT 2000*, *Lecture Notes in Computer Science*, 1976. pp. 489–502. Berlin/Heidelberg: Springer.

[11] Anderson, R. and M. Kuhn (1997). "Low Cost Attacks on Tamper Resistant Devices", *Security Protocols—5th International Workshop*, *Lecture Notes in Computer Science*, 1361. pp. 125–136. Berlin/Heidelberg: Springer.

[12] * Apostol, T. M. (1976). *Introduction to Analytic Number Theory*. *Undergraduate Texts in Mathematics*. New York: Springer.

[13] Arnold, V. I. (1999). "Polymathematics: Is Mathematics a Single Science or a Set of Arts?", in V. Arnold, M. Atiyah, P. Lax and B. Mazur (eds.), *Mathematics: Frontiers and Perspectives*, pp. 403–416. Providence, Rhode Island: American Mathematical Society.

[14] Atiyah, M. F. and I. G. MacDonald (1969). *Introduction to Commutative Algebra*. Reading, Massachusetts: Addison-Wesley.

[15] Aumüller, C., P. Bier, W. Fischer, P. Hofreiter and J.-P. Seifert (2002), "Fault Attacks on RSA with CRT: Concrete Results and Practical Countermeasures" [online document]. Available at http://eprint.iacr.org/2002/073 (October 2008).

[16] Balasubramanian, R. and N. Koblitz (1998). "The Improbability that an Elliptic Curve has Subexponential Discrete Log Problem under the Menezes-Okamoto Vanstone Algorithm", *Journal of Cryptology*, 11: 141–145.

[17] Bao, F., R. H. Deng, Y. Han, A. B. Jeng, A. D. Narasimhalu, T.-H. Ngair (1997). "Breaking Public Key Cryptosystems on Tamper Resistant Devices in the Presence of Transient Faults", *Security Protocols—5th International Workshop*, *Lecture Notes in Computer Science*, 1361. pp. 115–124. Berlin/Heidelberg: Springer.

[18] Bellare, M. and P. Rogaway (1995). "Optimal Asymmetric Encryption—How to Encrypt with RSA", *Advances in Cryptology—EUROCRYPT '94*, *Lecture Notes in Computer Science*, 950. pp. 92–111. Berlin/Heidelberg: Springer. A revised version is available at http://www-cse.ucsd.edu/users/mihir/papers/oaep.html (October 2008).

[19] ——— (1996). "The Exact Security of Digital Signatures: How to Sign with RSA and Rabin", *Advances in Cryptology—EUROCRYPT '96*, *Lecture Notes in Computer Science*, 1070. pp. 399–416. Berlin/Heidelberg: Springer. A revised version is available at http://www-cse.ucsd.edu/users/mihir/papers/exactsigs.html (October 2008).

[20] Bennett, C. H. and G. Brassard (1984). "Quantum Cryptography: Public Key Distribution and Coin Tossing", pp. 175–179. Proceedings of the IEEE International Conference on Computers, Systems and Signal Processing, Bangalore, India, December.

[21] Berlekamp, E. R. (1968). *Algebraic Coding Theory*. New York: McGraw-Hill.

[22] Biham, E. and A. Shamir (1997). "Differential Fault Analysis of Secret Key Cryptosystems", *Advances in Cryptology—CRYPTO '97*, *Lecture Notes in Computer Science*, 1294. pp. 513–528. Berlin/Heidelberg: Springer.

[23] Blake, I. F., R. Fuji-Hara, R. C. Mullin and S. A. Vanstone (1984). "Computing Logarithms in Finite Fields of Characteristic Two", *SIAM Journal of Algebraic and Discrete Methods*, 5: 276–285.

[24] Blake, I. F., G. Seroussi and N. P. Smart (1999). *Elliptic Curves in Cryptography*. Cambridge: Cambridge University Press.

[25] Blom, R. (1985). "An Optimal Class of Symmetric Key Generation Systems", *Advances in Cryptology—EUROCRYPT '84*, *Lecture Notes in Computer Science*, 209. pp. 335–338. Berlin/Heidelberg: Springer.

[26] Blum, L., M. Blum, and M. Shub (1986). "A Simple Unpredictable Pseudo-Random Number Generator", *SIAM Journal on Computing*, 15: 364–383.

[27] Blum, M. and S. Goldwasser (1985). "An Efficient Probabilistic Public Key Encryption Scheme Which Hides All Partial Information", *Advances in Cryptology—CRYPTO '84*, *Lecture Notes in Computer Science*, 196. pp. 289–299. Berlin/Heidelberg: Springer.

[28] Blundo, C., A. De Santis, A. Herzberg, S. Kutten, U. Vaccaro and M. Yung (1993). "Perfectly-Secure Key Distribution for Dynamic Conferences", *Advances in Cryptology—CRYPTO '92*, *Lecture Notes in Computer Science*, 740. pp. 471–486. Berlin/Heidelberg: Springer.

[29] Boneh, D. (1999). "Twenty Years of Attacks on the RSA Cryptosystem", *Notices of the American Mathematical Society*, 46 (2): 203–213.

[30] Boneh, D., R. A. DeMillo and R. J. Lipton (1997). "On the Importance of Checking Cryptographic Protocols for Faults", *Advances in Cryptology—EUROCRYPT '97*, *Lecture Notes in Computer Science*, 1233. pp. 37–51. Berlin/Heidelberg: Springer.

[31] ——— (2001). "On the Importance of Eliminating Errors in Cryptographic Computations", *Journal of Cryptology*, 14 (2): 101–119.

[32] Boneh, D. and G. Durfee (1999). "Cryptanalysis of RSA with Private Key d Less Than $N^{0.292}$", *Advances in Cryptology—EUROCRYPT '99*, *Lecture Notes in Computer Science*, 1592. pp. 1–11. Berlin/Heidelberg: Springer.

[33] Boneh, D., G. Durfee and Y. Frankel (1998). "Exposing an RSA Private Key Given a Small Fraction of Its Bits", *Advances in Cryptology—ASIACRYPT '98*, *Lecture Notes in Computer Science*, 1514. pp. 25–34. Berlin/Heidelberg: Springer.

[34] Boneh, D. and M. K. Franklin (2001). "Identity-based Encryption from the Weil Pairing", *Advances in Cryptology—CRYPTO 2001*, *Lecture Notes in Computer Science*, 2139. pp. 213–229. Berlin/Heidelberg: Springer.

[35] ——— (2003). "Identity-based Encryption from the Weil Pairing", *SIAM Journal of Computing*, (32) 3: 586–615.

[36] Bressoud, D. M. (1989). *Factorization and Primality Testing*. *Undergraduate Texts in Mathematics*. New York: Springer.

[37] * Buchmann, J. A. (2004). *Introduction to Cryptography*. *Undergraduate Texts in Mathematics*. New York: Springer.

[38] Buchmann, J. A. et al. (2004), "The Number Field Cryptography Project" [online document]. Available at http://www.informat k.tu-darmstadt.de/TI/Forschung/nfc.html (October 2008).

[39] Buchmann, J. A. and S. Hamdy (2001). "A Survey on IQ Cryptography". Technical report TI-4/01, TU Darmstadt, Fachbereich Informatik.

[40] Buchmann, J. A. and D. Weber (2000). "Discrete Logarithms: Recent Progress", in J. Buchmann, T. Hoeholdt, H. Stichtenoth and H. Tapia-Recillas (eds.), *Coding Theory, Cryptography and Related Areas*, pp. 42–56. Proceedings of an International Conference on Coding Theory, Cryptography and Related Areas, Guanajuato, Mexico, April 1998.

[41] Buhler, J., H. W. Lenstra and C. Pomerance (1993). "Factoring Integers with the Number Field Sieve", in A. K. Lenstra and H. W. Lenstra (eds.), *The Development of the Number Field Sieve*, *Lecture Notes in Mathematics*, 1554. pp. 50–94. Berlin: Springer.

[42] * Burton, D. M. (1998). *Elementary Number Theory*, 4th ed. New York: McGraw-Hill.

[43] Cantor, D. G. (1994). "On the Analogue of Division Polynomials for Hyperelliptic Curves", *Journal für die reine und angewandte Mathematik*, 447: 91–145.

[44] Chan, H., A. Perrig and D. Song (2003). "Random Key Predistribution Schemes for Sensor Networks", pp. 197–213. Proeedings of the 24th IEEE Symposium on Research in Security and Privacy, Berkeley, California, 11–14 May.

[45] Chari, S., C. S. Jutla, J. R. Rao, and P. Rohatgi (1999). "Towards Sound Approaches to Counteract Power-Analysis Attacks", *Advances in Cryptology—CRYPTO '99*, *Lecture Notes in Computer Science*, 1666. pp. 398–412. Berlin/Heidelberg: Springer.

[46] Charlap, L. S. and R. Coley (1990). "An Elementary Introduction to Elliptic Curves II", CCR Expository Report 34.

[47] Charlap, L. S. and D. P. Robbins (1988). "An Elementary Introduction to Elliptic Curves", CRD Expository Report 31.

[48] Chaum, D. (1983). "Blind Signatures for Untraceable Payments", *Advances in Cryptology—CRYPTO '82*. pp. 199–203. New York: Plenum Press.

[49] ——— (1985). "Security Without Identification: Transaction System to Make Big Brother Obsolete", *Communications of the ACM*, 28 (10): 1030–1044.

[50] ——— (1989). "Privacy Protected Payments: Unconditional Payer and/or Payee Untraceability", *Smart Card 2000: The Future of IC Cards*, pp. 69–93. Amsterdam: North-Holland.

[51] ——— (1990). "Zero-Knowledge Undeniable Signatures", *Advances in Cryptology—CRYPTO '90*, *Lecture Notes in Computer Science*, 473. pp. 458–464. Berlin/Heidelberg: Springer.

[52] Chaum, D. and H. van Antwerpen (1989). “Undeniable Signatures”, *Advances in Cryptology—CRYPTO ’89*, *Lecture Notes in Computer Science*, 435. pp. 212–217. Berlin/Heidelberg: Springer.

[53] Chaum, D., E. van Heijst and B. Pfitzmann (1991). “Cryptographically Strong Undeniable Signatures, Unconditionally Secure for the Signer”, *Advances in Cryptology—CRYPTO ’91*, *Lecture Notes in Computer Science*, 576. pp. 470–484. Berlin/Heidelberg: Springer.

[54] Chor, B. and R. L. Rivest (1988). “A Knapsack Type Cryptosystem Based on Arithmetic in Finite Fields”, *IEEE Transactions on Information Theory*, 34: 901–909.

[55] Clavier, C., J.-S. Coron and N. Dabbous (2000). “Differential Power Analysis in the Presence of Hardware Countermeasures”, *Cryptographic Hardware and Embedded Systems—CHES 2000*, *Lecture Notes in Computer Science*, 1965. pp. 252–263. Berlin/Heidelberg: Springer.

[56] Cohen, H. (1993). *A Course in Computational Algebraic Number Theory*. *Graduate Texts in Mathematics*, 138. New York: Springer.

[57] Coppersmith, D. (1984). “Fast Evaluation of Logarithms in Fields of Characteristic Two”, *IEEE Transactions on Information Theory*, 30: 587–594.

[58] ——— (1994). “Solving Homogeneous Equations over GF$[2]$ via Block Wiedemann Algorithm”, *Mathematics of Computation*, 62: 333–350.

[59] Coppersmith, D., A. M. Odlyzko and R. Schroeppel (1986). “Discrete Logarithms in $GF(p)$”, *Algorithmica*, 1: 1–15.

[60] Coppersmith, D. and S. Winograd (1982). “On the Asymptotic Complexity of Matrix Multiplication”, *SIAM Journal of Computing*, 11 (3): 472–492.

[61] * Cormen, T. H., C. E. Lieserson, R. L. Rivest and C. Stein (2001). *Introduction to Algorithms*, 2nd ed. Cambridge, Massachusetts: MIT Press.

[62] Coron, J.-S. (1999). “Resistance Against Differential Power Analysis for Elliptic Curve Cryptosystems”, *Cryptographic Hardware and Embedded Systems—CHES 1999*, *Lecture Notes in Computer Science*, 1965. pp. 292–302. Berlin/Heidelberg: Springer.

[63] Coron, J.-S., L. Goubin (2000). “On Boolean and Arithmetic Masking Against Differential Power Analysis”, *Cryptographic Hardware and Embedded Systems—CHES 2000*, *Lecture Notes in Computer Science*, 1965. pp. 231–237. Berlin/Heidelberg: Springer.

[64] Coster, M. J., A. Joux, B. A. LaMacchia, A. M. Odlyzko, C. P. Schnorr and J. Stern (1992). “Improved Low-Density Subset Sum Algorithms”, *Computational Complexity*, 2: 111–128.

[65] Coster, M. J., B. A. LaMacchia, A. M. Odlyzko and C. P. Schnorr (1991). “An Improved Low-Density Subset Sum Algorithm”, *Advances in Cryptology—EUROCRYPT ’91*, *Lecture Notes in Computer Science*, 547. pp. 54–67. Berlin/Heidelberg: Springer.

[66] Courtois, N. (2003). “Fast Algebraic Attacks on Stream Ciphers with Linear Feedback”, *Advances in Cryptology—CRYPTO 2003*, *Lecture Notes in Computer Science*, 2729. pp. 177–194. Berlin/Heidelberg: Springer.

[67] Courtois, N. and W. Meier (2003). “Algebraic Attacks on Stream Ciphers with Linear Feedback”, *Advances in Cryptology—EUROCRYPT 2003*, *Lecture Notes in Computer Science*, 2656. pp. 345–359. Berlin/Heidelberg: Springer.

[68] Courtois, N. and J. Pieprzyk (2003). “Cryptanalysis of Block Ciphers with Overdefined Systems of Equations”, *Advances in Cryptology—ASIACRYPT 2002*, *Lecture Notes in Computer Science*, 2501. pp. 267–287. Berlin/Heidelberg: Springer.

[69] Crandall, R. and C. Pomerance (2001). *Prime Numbers: A Computational Perspective*. New York: Springer.

[70] Crépeau, C. and A. Slakmon (2003). “Simple Backdoors for RSA Key Generation”, *Topics in Cryptology—CT-RSA 2003*, *Lecture Notes in Computer Science*, 2612. pp. 403–416. Berlin/Heidelberg: Springer.

[71] Daemen, J. and V. Rijmen (2002). *The Design of Rijndael: AES—The Advanced Encryption Standard*. New York: Springer.

[72] Das, A. (1999). Galois Field Computations: Implementation of a Library and a Study of the Discrete Logarithm Problem [dissertation]. Bangalore, India: Indian Institute of Science.

[73] Das, A. and C. E. Veni Madhavan (1999). "Performance Comparison of Linear Sieve and Cubic Sieve Algorithms for Discrete Logarithms over Prime Fields", *Algorithms and Computation, ISAAC '99, Lecture Notes in Computer Science*, 1741. pp. 295–306. Berlin/Heidelberg: Springer.

[74] * Delfs, H. and H. Knebl (2007). *Introduction to Cryptography: Principles and Applications*, 2nd ed. Berlin and New York: Springer.

[75] Deutsch, D. (1985). "Quantum Theory, the Church-Turing Principle and the Universal Quantum Computer". *Proceedings of the Royal Society of London, Series A*, 400. pp. 97–117.

[76] ——— (1998). *The Fabric of Reality: The Science of Parallel Universes—and Its Implications*. London: Penguin.

[77] Dhem, J.-F., F. Koeune, P.-A. Leroux, P. Mestré, J.-J. Quisquater and J.-L. Willems (2000). "A Practical Implementation of the Timing Attack", in J.-J. Quisquater and B. Schneier (eds.), *Smart Card: Research and Applications, Lecture Notes in Computer Science*, 1820. Proceedings of the Third Working Conference on Smart Card Research and Advanced Applications—CARDIS '98, Louvain-la-Neuve, Belgium, 14–16 September 1998. Springer.

[78] Diffie, W. and M. Hellman (1976). "New Directions in Cryptography", *IEEE Transactions on Information Theory*, 22: 644–654.

[79] Du, W., J. Deng, Y. S. Han and P. K. Varshney (2003). "Establishing Pairwise Keys in Distributed Sensor Networks", pp. 42–51. Proceedings of the 10th ACM Conference on Computer and Communication Security, Washington D.C., USA, 27–30 October.

[80] Du, W., J. Deng, Y. S. Han, S. Chen and P. K.Varshney (2004). "A Key Management Scheme for Wireless Sensor Networks Using Deployment Knowledge". Proceedings of IEEE INFOCOM 2004, Hong Kong, 7–11 March.

[81] * Dummit, D. and R. Foote (2004). *Abstract Algebra*, 3rd ed. Somerset, New Jersey: John Wiley & Sons.

[82] Durfee, G. and P. Q. Nguyen (2000). "Cryptanalysis of the RSA Schemes with Short Secret Exponent from Asiacrypt '99", *Advances in Cryptology—ASIACRYPT 2000, Lecture Notes in Computer Science*, 1976. pp. 30–44. Berlin/Heidelberg: Springer.

[83] Dusart, P. (1999). "The kth Prime Is Greater than $k(\ln k+\ln\ln k-1)$ for $k > 2$", *Mathematics of Computation*, 68: 411–415.

[84] ElGamal, T. (1985). "A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms", *IEEE Transactions on Information Theory*, 31: 469–472.

[85] Elkies, N. D. (1998). "Elliptic and Modular Curves over Finite Fields and Related Computational Issues", *AMS/IP Studies in Advanced Mathematics*, 7: 21–76.

[86] Enge, A. (1999). "Computing Discrete Logarithms in High-Genus Hyperelliptic Jacobians in Provably Subexponential Time". Technical report CORR 99-04, University of Waterloo, Canada.

[87] Enge, A. and P. Gaudry (2002). "A General Framework for Subexponential Discrete Logarithm Algorithms", *Acta Arithmetica*, 102 (1): 83–103.

[88] Eschenauer, L. and V. D. Gligor (2002). "A Key-Management Scheme for Distributed Sensor Networks". Proceedings of the 9th ACM Conference on Computer and Communication Security, pp. 41–47. Washington D.C., USA, 18–22 November.

[89] * Esmonde, J. and M. Ram Murty (1999). *Problems in Algebraic Number Theory. Graduate Texts in Mathematics*, 190. New York: Springer.

[90] Fiat, A. and A. Shamir (1987). “How to Prove Yourself: Practical Solutions to Identification and Signature Problems”, *Advances in Cryptology—CRYPTO ’86*, *Lecture Notes in Computer Science*, 263. pp. 186–194. Berlin/Heidelberg: Springer.

[91] Feige, U., A. Fiat, and A. Shamir (1988). “Zero-Knowledge Proofs of Identity”, *Journal of Cryptology*, 1: 77–94.

[92] * Feller, W. (1966). *Introduction to Probability Theory and Its Applications*, 3rd ed. New York: John Wiley & Sons.

[93] Ferguson, N., J. Kelsey, S. Lucks, B. Schneier, M. Stay, D. Wagner and D. Whiting (2000). “Improved Cryptanalysis of Rijndael”, *Fast Software Encryption—FSE 2000*, *Lecture Notes in Computer Science*, 1978. pp. 213–230. Berlin/Heidelberg: Springer.

[94] Fouquet, M., P. Gaudry and R. Harley (2000). “An Extension of Satoh’s Algorithm and Its Implementation”, *Journal of Ramanujan Mathematical Society*, 15: 281–318.

[95] ——— (2001). “Finding Secure Curves with the Satoh-FGH Algorithm and an Early-Abort Strategy”, *Advances in Cryptology—EUROCRYPT 2001*, *Lecture Notes in Computer Science*, 2045. Berlin/Heidelberg: Springer.

[96] * Fraleigh, J. B. (1998). *A First Course in Abstract Algebra*, 6th ed. Reading, Massachusetts: Addison-Wesley.

[97] Fujisaki, E., T. Kobayashi, H. Morita, H. Oguro, T. Okamoto, S. Okazaki, D. Pointcheval and S. Uchiyama (1999). “EPOC: Efficient Probabilistic Public-Key Encryption”, contribution to IEEE P1363a.

[98] Fujisaki, E., T. Okamoto, D. Pointcheval, J. Stern (2001). “RSA-OAEP is Secure under the RSA Assumption”, *Advances in Cryptology—CRYPTO 2001*, *Lecture Notes in Computer Science*, 2139. pp. 260–274. Berlin/Heidelberg: Springer.

[99] Fulton, W. (1969). *Algebraic Curves*. *Mathematics Lecture Notes Series*. New York: W. A. Benjamin.

[100] Galbraith, S. D. (2003). “Weil Descent of Jacobians”, *Discrete Applied Mathematics*, 128 (1): 165–180.

[101] Galbraith, S. D., F. Hess and N. P. Smart (2002). “Extending the GHS Weil Descent Attack”, *Advances in Cryptology—EUROCRYPT 2002*, *Lecture Notes in Computer Science*, 2332. pp. 29–44. Berlin/Heidelberg: Springer.

[102] Galbraith, S. D., W. Mao, and K. G. Paterson (2002). “RSA-based Undeniable Signatures for General Moduli”, *Topics in Cryptology—CT-RSA 2002*, *Lecture Notes in Computer Science*, 2271. pp. 200–217. Berlin/Heidelberg: Springer.

[103] Gathen, J. von zur and J. Gerhard (1999). *Modern Computer Algebra*. Cambridge: Cambridge University Press.

[104] Gathen, J. von zur and V. Shoup (1992). “Computing Frobenius Maps and Factoring Polynomials”, pp. 97–105. Proceedings of the 24th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada.

[105] Gaudry, P. (2000). “An Algorithm for Solving the Discrete Log Problem on Hyperelliptic Curves”, *Advances in Cryptology—EUROCRYPT 2000*, *Lecture Notes in Computer Science*, 1807. pp. 19–34. Berlin/Heidelberg: Springer.

[106] Gaudry, P. and R. Harley (2000). “Counting Points on Hyperelliptic Curves over Finite Fields”, *Algorithmic Number Theory—ANTS-IV*, *Lecture Notes in Computer Science*, 1838. pp. 313–332. Berlin/Heidelberg: Springer.

[107] Gaudry, P., F. Hess and N. P. Smart (2002). “Constructive and Destructive Facets of Weil Descent on Elliptic Curves”, *Journal of Cryptology*, 15 (1): 19–46.

[108] Geddes, K. O., S. R. Czapor and G. Labahn (1992). *Algorithms for Computer Algebra*. Boston: Kluwer Academic Publishers.

[109] Gennaro, R., H. Krawczyk and T. Rabin (2000). “RSA-based Undeniable Signatures”, *Journal of Cryptology*, 13 (4): 397–416.

[110] Gentry, C., J. Jonsson, M. Szydlo and J. Stern (2001). "Cryptanalysis of the NTRU Signature Scheme (NSS) from Eurocrypt 2001", *Advances in Cryptology—ASIACRYPT 2001*, *Lecture Notes in Computer Science*, 2248. pp. 1–20. Berlin/Heidelberg: Springer.

[111] Gentry, C. and M. Szydlo (2002). "Cryptanalysis of the NTRU Signature Scheme", *Advances in Cryptology—EUROCRYPT '02*, *Lecture Notes in Computer Science*, 2332. pp. 299–320. Berlin/Heidelberg: Springer.

[112] Gilbert, H. and M. Minier (2000). "A Collision Attack on Seven Rounds of Rijndael", pp. 230–241. Proceedings of the 3rd AES Conference, NIST, New York, April 2000.

[113] * Goldreich, O. (2001). *Foundations of Cryptography*, *Volume 1: Basic Tools*. Cambridge: Cambridge University Press.

[114] * ——— (2004). *Foundations of Cryptography*, *Volume 2: Basic Applications*. Cambridge: Cambridge University Press.

[115] Goldreich, O., S. Goldwasser and S. Halevi (1997). "Public-key Cryptosystems from Lattice Reduction Problems", *Advances in Cryptology—CRYPTO '97*, *Lecture Notes in Computer Science*, 1294. pp. 112–131. Berlin/Heidelberg: Springer.

[116] Goldwasser, S. and J. Kilian (1986). "Almost All Primes Can Be Quickly Certified", pp. 316–329. Prodeedings of the 18th Annual ACM Symposium on Theory of Computing, Berkeley, California.

[117] Goldwasser, S. and S. Micali (1984). "Probabilistic Encryption", *Journal of Computer and Systems Sciences*, 28: 270–299.

[118] Gordon, D. M. (1985). "Strong Primes are Easy to Find", *Advances in Cryptology—EUROCRYPT '84*, *Lecture Notes in Computer Science*, 209. pp. 216–223. Berlin/Heidelberg: Springer.

[119] ——— (1993). "Discrete Logarithms in $GF(p)$ Using the Number Field Sieve", *SIAM Journal of Discrete Mathematics*, 6: 124–138.

[120] Gordon, D. M. and K. S. McCurley (1992). "Massively Parallel Computation of Discrete Logarithms", *Advances in Cryptology—CRYPTO '92*, *Lecture Notes in Computer Science*, 740. pp. 312–323. Berlin/Heidelberg: Springer.

[121] Grinstead, C. M. and J. L. Snell (1997). *Introduction to Probability*, 2nd revised ed. Providence, Rhode Island: American Mathematical Society. The book is also available at http://www.dartmouth.edu/∼chance/book.html (October 2008).

[122] Guillou, L. C. and J.-J. Quisquater (1988). "A Practical Zero-Knowledge Protocol Fitted to Security Microprocessor Minimizing Both Trasmission and Memory", *Advances in Cryptology—EUROCRYPT '88*, *Lecture Notes in Computer Science*, 330. pp. 123–128. Berlin/Heidelberg: Springer.

[123] Hankerson, D., A. J. Menezes and S. Vanstone (2004). *Guide to Elliptic Curve Cryptography*. New York: Springer.

[124] Hartshorne, R. (1977). *Algebraic Geometry*. *Graduate Texts in Mathematics*, 52. New York, Heidelberg and Berlin: Springer.

[125] * Herstein, I. N. (1975). *Topics in Algebra*. New York: John Wiley & Sons.

[126] Hess, F., G. Seroussi and N. P. Smart (2000). "Two Topics in Hyperelliptic Cryptography". HP Labs technical report HPL-2000-118.

[127] * Hoffman, K. and R. Kunze (1971). *Linear Algebra*. Englewood Cliffs, New Jersey: Prentice-Hall.

[128] Hoffstein, J., N. Howgrave-Graham, J. Pipher, J. H. Silverman and W. White (2003). "NTRUSign: Digital Signatures Using the NTRU Lattice", *Topics in Cryptology—CT-RSA 2003*, *Lecture Notes in Computer Science*, 2612. pp. 122–140. Berlin/Heidelberg: Springer.

[129] ——— (2005). "Performance Improvements and a Baseline Parameter Generation Algorithm for NTRUSign", Workshop on Mathematical Problems and Techniques in Cryptology, Barcelona, Spain, June 2005. Also available at http://www. ntru.com/cryptolab/articles.htm (October 2008).

[130] Hoffstein, J., J. Pipher and J. H. Silverman (1998). “NTRU: A Ring-Based Public Key Cryptosystem”, *Algorithmic Number Theory—ANTS-III*, *Lecture Notes in Computer Science*, 1423. pp. 267–288. Berlin/Heidelberg: Springer.

[131] ——— (2001). “NSS: An NTRU Lattice-Based Signature Scheme”, *Advances in Cryptology—EUROCRYPT 2001*, *Lecture Notes in Computer Science*, 2045. pp. 211–228. Berlin/Heidelberg: Springer.

[132] Horster, P., M. Michels and H. Petersen (1994). “Meta-ElGamal Signature Schemes”. Technical report TR-94-5-F, Department of Computer Science, Teschnische Universität, Chemnitz-Zwickau.

[133] * Hungerford, T. W. (1974). *Algebra*, 5th ed. *Graduate Texts in Mathematics*, 73. Berlin: Springer.

[134] IEEE (2008), “Standard Specifications for Public-Key Cryptography” [online document]. Available at http://grouper.ieee.org/groups/1363/index.html (October 2008).

[135] IETF (2008), “The Internet Engineering Task Force” [online document]. Available at http://www.ietf.org/ (October 2008).

[136] * Ireland, K. and M. Rosen (1990). *A Classical Introduction to Modern Number Theory*. *Graduate Texts in Mathematics*, 84. New York: Springer.

[137] Izu, T., B. Möller and T. Takagi (2002). “Improved Elliptic Curve Multiplication Methods Resistant Against Side Channel Attacks”, *Progress in Cryptology—INDOCRYPT 2002*, *Lecture Notes in Computer Science*, 2551. pp. 296–313. Berlin/Heidelberg: Springer.

[138] Izu, T. and T. Takagi (2002). “A Fast Parallel Elliptic Curve Multiplication Resistant Against Side Channel Attacks”, *Public Key Cryptography—PKC 2002*, *Lecture Notes in Computer Science*, 2274. pp. 280–296. Berlin/Heidelberg: Springer. An improved version of this paper is published as the technical report CORR 2002-03 of the Centre for Applied Cryptographic Research, University of Waterloo, Canada, and is available at http://www.cacr.math.uwaterloo.ca/ (October 2008).

[139] Jacobson, M. J., N. Koblitz, J. H. Silverman, A. Stein and E. Teske (2000). “Analysis of the Xedni Calculus Attack”, *Design, Codes and Cryptography*, 20: 41–64.

[140] Janusz, G. J. (1995). *Algebraic Number Fields*. Providence, Rhode Island: American Mathematical Society.

[141] Johnson, D. and A. Menezes (1999). “The Elliptic Curve Digitial Signature Algorithm (ECDSA)”. Technical report CORR 99-34, Department of Combinatorics and Optimization, University of Waterloo, Canada. Also published in *International Journal on Information Security* (2001), 1: 36–63.

[142] Joye, M., A. K. Lenstra and J.-J. Quisquater (1999). “Chinese Remaindering Based Cryptosystems in the Presence of Faults”, *Journal of Cryptology*, 12 (4): 241–246.

[143] Kaltofen, E. and V. Shoup (1995). “Subquadratic-Time Factoring of Polynomials over Finite Fields”, pp. 398–406. Proceedings of the 27th Annual ACM Symposium on Theory of Computing, Las Vegas, Nevada.

[144] Kampkötter, W. (1991). Explizite Gleichungen für Jacobishe Varietäten hyperelliptischer Kurven [dissertation]. Essen: Gesamthochschule.

[145] Katz, J. and Y. Lindell (2007). *Introduction to Modern Cryptography*. Boca Raton, Florida; London and New York: CRC Press.

[146] Kaye, P. and C. Zalka (2004), “Optimized Quantum Implementation of Elliptic Curve Arithmetic over Binary Fields” [online document]. Available at http://arxiv.org/abs/quant-ph/0407095 (October 2008).

[147] * Knuth, D. E. (1997). *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*. Reading, Massachusetts: Addison-Wesley.

[148] Ko, K. H., S. J. Lee, J. H. Cheon, J. W. Han, J. S. Kang and C. S. Park (2000). “New Public-Key Cryptosystem Using Braid Groups”, *Advances in Cryptology—CRYPTO 2000*, *Lecture Notes in Computer Science*, 1880. pp. 166–183. Berlin/Heidelberg: Springer.

[149] Koblitz, N. (1984). *p-adic Numbers, p-adic Analysis, and Zeta-Functions*, 2nd ed. *Graduate Texts in Mathematics*, 58. New York, Heidelberg and Berlin: Springer.

[150] ——— (1987). “Elliptic Curve Cryptosystems”, *Mathematics of Computation*, 48: 203–209.

[151] ——— (1989). “Hyperelliptic Cryptosystems”, *Journal of Cryptology*, 1: 139–150.

[152] ——— (1993). *Introduction to Elliptic Curves and Modular Forms*, 2nd ed. *Graduate Texts in Mathematics*, 97. Berlin: Springer.

[153] * ——— (1994). *A Course in Number Theory and Cryptography*, 2nd ed. New York:Springer.

[154] ——— (1998). *Algebraic Aspects of Cryptography*. New York: Springer.

[155] Kocher, P. C. (1996). “Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems”, *Advances in Cryptology—CRYPTO ’96*, *Lecture Notes in Computer Science*, 1109. pp. 104–113. Berlin/Heidelberg: Springer.

[156] Kocher, P. C., J. Jaffe and B. Jun (1999). “Differential Power Analysis”, *Advances in Cryptology—CRYPTO ’99*, *Lecture Notes in Computer Science*, 1666. pp. 388–397. Berlin/Heidelberg: Springer.

[157] Lagarias, J. C. and A. M. Odlyzko (1985). “Solving Low-Density Subset Sum Problems”, *Journal of ACM*, 32: 229–246.

[158] LaMacchia, B. A. and A. M. Odlyzko (1991a). “Computation of Discrete Logarithms in Prime Fields”, *Designs, Codes and Cryptography*, 1: 46–62.

[159] ——— (1991b). “Solving Large Sparse Linear Systems over Finite Fields”, *Advances in Cryptology—CRYPTO ’90*, *Lecture Notes in Computer Science*, 537. pp. 109–133. Berlin/Heidelberg: Springer.

[160] Lang, S. (1994). *Algebraic Number Theory*. *Graduate Texts in Mathematics*, 110. New York: Springer.

[161] Law, L., A. Menezes, A. Qu, J. Solinas and S. Vanstone (1998). “An Efficient Protocol for Authenticated Key Agreement”. Technical report CORR 98-05, Department of Combinatorics and Optimization, University of Waterloo, Canada.

[162] Lehmer, D. H. and R. E. Powers (1931). “On Factoring Large Numbers”, *Bulletin of the AMS*, 37: 770–776.

[163] Lenstra, A. K., E. Tromer, A. Shamir, W. Kortsmit, B. Dodson, J. Hughes and P. Leyland (2003). “Factoring Estimates for a 1024-Bit RSA Modulus”, *Advances in Cryptology—ASIACRYPT 2003*, *Lecture Notes in Computer Science*, 2894. pp. 55–74. Berlin/Heidelberg: Springer.

[164] Lenstra, A. K. and H. W. Lenstra (1990). “Algorithms in Number Theory”, in J. van Leeuwen (ed.), *Handbook of Theoretical Computer Science*, *Volume A*, pp. 675–715, Amsterdam: Elsevier.

[165] Lenstra, A. K. and H. W. Lenstra (ed.) (1993). *The Development of the Number Field Sieve*. *Lecture Notes in Mathematics*, 1554. Berlin: Springer.

[166] Lenstra, A. K., H. W. Lenstra and L. Lovasz (1982). “Factoring Polynomials with Rational Coefficients”, *Mathematische Annalen*, 261: 515–534.

[167] Lenstra, A. K., H. W. Lenstra, M. S. Manasse and J. M. Pollard (1990). “The Number Field Sieve”, pp. 564–572. Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, Baltimore, Maryland, USA, 13–17 May.

[168] Lenstra, A. K. and A. Shamir (2000). “Analysis and Optimization of the TWINKLE Factoring Device”, *Advances in Cryptology—EUROCRYPT 2000*, *Lecture Notes in Computer Science*, 1807. pp. 35–52. Berlin/Heidelberg: Springer.

[169] Lenstra, A. K., A. Shamir, J. Tomlinson and E. Tromer (2002). “Analysis of Bernstein’s Factorization Circuit”, *Advances in Cryptology—ASIACRYPT 2002*, *Lecture Notes in Computer Science*, 2501. pp. 1–26. Berlin/Heidelberg: Springer.

[170] Lenstra, A. K. and E. R. Verheul (2000a). “The XTR Public Key System”, *Advances in Cryptology—CRYPTO 2000*, *Lecture Notes in Computer Science*, 1880. pp. 1–20. Berlin/Heidelberg: Springer.

[171] ——— (2000b). “Key Improvements to XTR”, *Advances in Cryptology—ASIACRYPT 2000*, *Lecture Notes in Computer Science*, 1976. pp. 220–233. Berlin/Heidelberg: Springer.

[172] ——— (2001a). “An Overview of the XTR Public Key System”, pp. 151–180. Proceedings of the Public Key Cryptography and Computational Number Theory Conference, Warsaw, Poland, 2000. Berlin: Verlages Walter de Gruyter.

[173] ——— (2001b). “Fast Irreducibility and Subgroup Membership Testing in XTR”, *Public Key Cryptography—PKC 2001, Lecture Notes in Computer Science*, 1992. pp. 73–86. Berlin/Heidelberg: Springer.

[174] Lenstra, H. W. (1987). “Factoring Integers with Elliptic Curves”, *Annals of Mathematics*, 126: 649–673.

[175] Lenstra, H. W. and C. Pomerance (2005), “Primality Testing with Gaussian Periods” [online document]. Available at http://www.math.dartmouth.edu/∼carlp/PDF/complexity12.pdf (October 2008).

[176] Lercier, R. (1997). “Finding Good Random Elliptic Curves for Cryptosystems Defined over $\mathbb{F}_{2^n}$”, *Advances in Cryptology—EUROCRYPT ’97, Lecture Notes in Computer Science*, 1233. pp. 379–392. Berlin/Heidelberg: Springer.

[177] Lercier, R. and D. Lubicz (2003). “Counting Points on Elliptic Curves over Finite Fields of Small Characteristic in Quasi Quadratic Time”, *Advances in Cryptology—EUROCRYPT 2003, Lecture Notes in Computer Science*, 2656. pp. 360–373. Berlin/Heidelberg: Springer.

[178] Libert, B. and J.-J. Quisquater (2003), “New Identity Based Signcryption Schemes from Pairings” [online document]. Available at http://eprint.iacr.org/2003/023/ (October 2008).

[179] Lidl, R. and H. Niederreiter (1984). *Finite Fields*, *Encyclopedia of Mathematics and Its Applications*, 20. Cambridge: Cambridge University Press.

[180] ——— (1994). *Introduction to Finite Fields and Their Applications*. Cambridge: Cambridge University Press.

[181] Liu, D. and P. Ning (2003a). “Establishing Pairwise Keys in Distributed Sensor Networks”, pp. 52–61. Proceedings of the 10th ACM Conference on Computer and Communication Security, Washington D.C., USA, October 2003.

[182] ——— (2003b). “Location-Based Pairwise Key Establishments for Static Sensor Networks”, pp. 72–82. Proceedings of the 1st ACM Workshop on Security in Ad Hoc and Sensor Networks, Fairfax, Virginia, 31 October 2003.

[183] Liu, D., P. Ning and R. Li (2005). “Establishing Pairwise Keys in Distributed Sensor Networks”, *ACM Transactions on Information and System Security*, (8) 1: 41–77.

[184] Lucks, S. (2000). “Attacking Seven Rounds of Rijndael Under 192-bit and 256-bit Keys”, pp. 215–229. Proceedings of the 3rd Advanced Encryption Standard Candidate conference, New York, April 2000.

[185] Malone-Lee, J. (2002), “Identity-Based Signcryption” [online document]. Available at http://eprint.iacr.org/2002/098/ (October 2008).

[186] Mao, W. (2001). “New Zero-Knowledge Undeniable Signatures—Forgery of Signature Equivalent to Factorisation”. Hewlett-Packard technical report HPL-2201-36.

[187] Mao, W. and K. G. Paterson (2000). “Convertible Undeniable Standard RSA Signatures”. Hewlett-Packard technical report HPL-2000-148.

[188] Matsumoto, T. and H. Imai (1988). “Public Quadratic Polynomial-Tuples for Efficient Signature-Verification and Message-Encryption”, *Advances in Cryptology—EUROCRYPT ’88, Lecture Notes in Computer Science*, 330. pp. 419–453. Berlin/Heidelberg: Springer.

[189] McCurley, K. S. (1990). “The Discrete Logarithm Problem”, in C. Pomerance and S. Goldwasser (eds.), *Cryptology and Computational Number Theory: American Mathematical Society Short Course*, Boulder, Colorado, 6–7 August 1989. Proceedings of Symposia in Applied Mathematics, 42. pp. 49–74. Providence, Rhode Island: American Mathematical Society.

[190] McEliece, R. J. (1978). “A Public-Key Cryptosystem Based on Algebraic Coding Theory”. DSN progress report 42–44, Jet Propulsion Laboratory, California Institute of Technology, pp. 114–116.

[191] Menezes, A. J. (ed.) (1993). *Applications of Finite Fields*. Boston: Kluwer Academic Publishers.

[192] Menezes, A. J. (1993). *Elliptic Curve Public Key Cryptosystems. The Springer International Series in Engineering and Computer Science*, 234. Springer. Available at http://books.google.co.in/books?id=bIb54ShKS68C (October 2008).

[193] Menezes, A. J., T. Okamoto and S. Vanstone (1993). "Reducing Elliptic Curve Logarithms to a Finite Field", *IEEE Transactions on Information Theory*, 39: 1639–1646.

[194] Menezes, A. J., P. van Oorschot and S. Vanstone (1997). *Handbook of Applied Cryptography*. Boca Raton, Florida: CRC Press.

[195] Menezes, A. J., Y. Wu and R. Zuccherato (1996). "An Elementary Introduction to Hyperelliptic Curves". CACR technical report CORR 96-19, University of Waterloo, Canada.

[196] Merkle, R. C. amd M. E. Hellman (1978). "Hiding Information and Signatures in Trapdoor Knapsacks", *IEEE Transactions on Information Theory*, 24 (5): 525–530.

[197] Mermin, N. D. (2003). "From Cbits to Qbits: Teaching Computer Scientists Quantum Mechanics", *American Journal of Physics*, 71: 23–30.

[198] Mermin, N. D. (2006), "Phys481-681-CS483 Lecture Notes and Homework Assignments" [online document]. Available at http://people.ccmr.cornell.edu/∼mermin/qcomp/CS483.html (October 2008).

[199] Messerges, T. S. (2000). "Securing the AES Finalists Against Power Analysis Attacks", *Fast Software Encryption—FSE 2000*, *Lecture Notes in Computer Science*, 1978. pp. 150–164. Berlin/Heidelberg: Springer.

[200] Messerges, T. S., E. A. Dabbish and R. H. Sloan (1999). "Power Analysis Attacks of Modular Exponentiation in Smartcards", *Cryptographic Hardware and Embedded Systems—CHES 1999*, *Lecture Notes in Computer Science*, 1717. pp. 144–157. Berlin/Heidelberg: Springer.

[201] ——— (2002). "Examining Smart-Card Security Under the Threat of Power Analysis Attacks", *IEEE Transactions on Computers*, 51 (4): 541–552.

[202] Michels, M. and M. Stadler (1997). "Efficient Convertible Undeniable Signature Schemes", pp. 231–244. Proceedings of the 4th International Workshop on Selected Areas in Cryptography, Ottawa, Canada.

[203] Mignotte, M. (1992). *Mathematics for Computer Algebra*. New York: Springer.

[204] Miller, G. L. (1976). "Riemann's Hypothesis and Tests for Primality", *Journal of Computer and System Sciences*, 13: 300–317.

[205] Miller, V. (1986). "Uses of Elliptic Curves in Cryptography", *Advances in Cryptology—CRYPTO '85*, *Lecture Notes in Computer Science*, 18. pp. 417–426. Berlin/Heidelberg: Springer.

[206] Möller, B. (2001). "Securing Elliptic Curve Point Multiplication Against Side-Channel Attacks", *Information Security Conference*, *Lecture Notes in Computer Science*, 2200. pp. 324–334. Berlin/Heidelberg: Springer.

[207] Mollin, R. A. (1998). *Fundamental Number Theory with Applications*. Boca Raton, Florida: Chapman & Hall/CRC.

[208] ——— (1999). *Algebraic Number Theory*. Boca Raton, Florida: Chapman & Hall/CRC.

[209] ——— (2001). *An Introduction to Cryptography*. Boca Raton, Florida: Chapman & Hall/CRC.

[210] Montgomery, P. L. (1985). "Modular Multiplication Without Trial Division", *Mathematics of Computation*, 44: 519–521.

[211] ——— (1994). "A Survey of Modern Integer Factorization Algorithms", *CWI Quarterly*, 7 (4): 337–366.

[212] ——— (1995). "A Block Lanczos Algorithm for Finding Dependencies over GF(2)", *Advances in Cryptology—EUROCRYPT '95*, *Lecture Notes in Computer Science*, 921. pp. 106–120. Berlin/Heidelberg: Springer.

[213] Morrison, M. A. and J. Brillhart (1975). "A Method of Factoring and a Factorization of F_7", *Mathematics of Computation*, 29: 183–205.

[214] * Motwani, R. and P. Raghavan (1995). *Randomized Algorithms*. Cambridge: Cambridge University Press.

[215] Muir, J. A. (2001). Techniques of Side Channel Cryptanalysis [dissertation]. Canada: University of Waterloo. Available at http://www.uwspace.uwaterloo.ca/bitstream/10012/1098/1/jamuir2001.pdf (October 2008).

[216] Neukirch, J. (1999). *Algebraic Number Theory*. Berlin and Heidelberg: Springer.

[217] Nguyen, P. Q. (2006), "A Note on the Security of NTRUSign" [online document]. Available at http://eprint.iacr.org/2006/387 (October 2008).

[218] * Nielsen, M. A. and I. L. Chuang (2000). *Quantum Computation and Quantum Information*. Cambridge: Cambridge University Press.

[219] NIST (2001), "Advanced Encryption Standard" [online document]. Available at http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf (October 2008).

[220] ——— (2006), "Digital Signature Standard (DSS)" [online document]. Available at http://csrc.nist.gov/publications/drafts/fips_186-3/Draft-FIPS-186-3%20_March2006.pdf (October 2008).

[221] ——— (2007a), "Federal Information Processing Standards" [online document]. Available at http://csrc.nist.gov/publications/PubsFIPS.html (October 2008).

[222] ——— (2007b), "Secure Hash Standard (SHS)" [online document]. Available at http://csrc.nist.gov/publications/drafts/fips_180-3/draft_fips-180-3_June-08-2007.pdf (October 2008).

[223] Nyberg, K. and R. A. Rueppel (1993). "A New Signature Scheme Based on the DSA Giving Message Recovery", pp. 58–61. Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, 3–5 November.

[224] ——— (1995). "Message Recovery for Signature Schemes Based on the Discrete Logarithm Problem", *Advances in Cryptology—EUROCRYPT '94*, *Lecture Notes in Computer Science*, 950. pp. 182–193. Berlin/Heidelberg: Springer.

[225] Odlyzko, A. M. (1985). "Discrete Logarithms and Their Cryptographic Significance", *Advances in Cryptology—EUROCRYPT '84*, *Lecture Notes in Computer Science*, 209. pp. 224–314. Berlin/Heidelberg: Springer.

[226] ——— (2000). "Discrete Logarithms: The Past and the Future", *Designs, Codes and Cryptography*, 19: 129–145.

[227] Okamoto, T. (1992). "Provably Secure and Practical Identification Schemes and Corresponding Signature Schemes", *Advances in Cryptology—CRYPTO '92*, *Lecture Notes in Computer Science*, 740. pp. 31–53. Berlin/Heidelberg: Springer.

[228] Okamoto, T., E. Fujisaki and H. Morita (1998). "TSH-ESIGN: Efficient Digital Signature Scheme Using Trisection Size Hash", submission to IEEE P1363a.

[229] Papadimitriou, C. H. (1994). *Computational Complexity*. Reading, Massachusetts: Addison-Wesley.

[230] Park, S., T. Kim, Y. An and D. Won (1995). "A Provably Entrusted Undeniable Signature", pp. 644–648. IEEE Singapore International Conference on Network/International Conference on Information Engineering (SICON/ICIE '95).

[231] Patarin, J. (1995). "Cryptanalysis of the Matsumoto and Imai Public Key Scheme of Eurocrypt'88", *Advances in Cryptology—CRYPTO '95*, *Lecture Notes in Computer Science*, 963. pp. 248–261. Berlin/Heidelberg: Springer.

[232] ——— (1996). "Hidden Fields Equations (HFE) and Isomorphisms of Polynomials (IP): Two New Families of Asymmetric Algorithms", *Advances in Cryptology—EUROCRYPT '96*, *Lecture Notes in Computer Science*, 1070. pp. 33–48. Berlin/Heidelberg: Springer.

[233] Pirsig, R. M. (1974). *Zen and the Art of Motorcycle Maintenance: An Inquiry into Values*. London: Bodley Head.

[234] Pohlig, S. and M. Hellman (1978). "An Improved Algorithm for Computing Logarithms over $GF(p)$ and its Cryptographic Significance", *IEEE Transactions on Information Theory*, 24: 106–110.

[235] Pohst, M. and H. Zassenhaus (1989). *Algorithmic Algebraic Number Theory*, *Encyclopaedia of Mathematics and Its Applications*, 30. Cambridge: Cambridge University Press.

[236] Pointcheval, D. and J. Stern (1996). "Provably Secure Blind Signature Schemes", *Advances in Cryptology—ASIACRYPT '96*, *Lecture Notes in Computer Science*, 1163. pp. 252–265. Berlin/Heidelberg: Springer.

[237] ——— (2000). "Security Arguments for Digital Signatures and Blind Signatures", *Journal of Cryptology*, 13 (3): 361–396.

[238] Pollard, J. M. (1974). "Theorems on Factorization and Primality Testing", *Proceedings of the Cambridge Philosophical Society*, 76 (2): 521–528.

[239] ——— (1975). "A Monte Carlo Method for Factorization", *BIT*, 15 (3): 331–334.

[240] ——— (1993). "Factoring with Cubic Integers", in A. K. Lenstra and H. W. Lenstra (eds.), *The Development of the Number Field Sieve*, *Lecture Notes in Mathematics*, 1554. pp. 4–10. Berlin: Springer.

[241] Pomerance, C. (1985). "The Quadratic Sieve Factoring Algorithm", *Advances in Cryptology—EUROCRYPT '84*, *Lecture Notes in Computer Science*, 209. pp. 169–182. Berlin/Heidelberg: Springer.

[242] ——— (2008). "Elementary Thoughts on Discrete Logarithms", pp. 385–396. in J. P. Buhler and P. Stevenhagen (eds.), *Surveys in Algorithmic Number Theory*, Publications of the Research Institute for Mathematical Sciences, 44. New York: Cambridge University Press.

[243] Preskill, J. (1998). "Quantum Computing: Pro and Con", *Proceedings of the Royal Society of London*, A454: 469–486.

[244] ——— (2007), "Course Information for Quantum Computation" [online document]. Available at http://theory.caltech.edu/people/preskill/ph219/ (October 2008).

[245] Proos, J. and C. Zalka (2004), "Shor's Discrete Logarithm Quantum Algorithm for Elliptic Curves" [online document]. Available at http://arxiv.org/abs/quant-ph/0301141 (October 2008).

[246] Rabin, M. O. (1979). "Digitalized Signatures and Public-Key Functions as Intractable as Factorization". Technical report MIT/LCS/TR-212, MIT Laboratory for Computer Science, Massachusetts.

[247] ——— (1980a). "Probabilistic Algorithms in Finite Fields", *SIAM Journal of Computing*, 9: 273–280.

[248] ——— (1980b). "Probabilistic Algorithm for Testing Primality", *Journal of Number Theory*, 12: 128–138.

[249] Ram Murty, M. (2001). *Problems in Analytic Number Theory*. New York: Springer.

[250] Raymond, J.-F. and A. Stiglic (2000), "Security Issues in the Diffie-Hellman Key Agreement Protocol" [online document]. Available at http://crypto.cs.mcgill.ca/∼stiglic/Papers/dhfull.pdf (October 2008).

[251] Ribenboim, P. (2001). *Classical Theory of Algebraic Numbers*. *Universitext*. New York: Springer.

[252] Rivest, R. L., A. Shamir, and L. M. Adleman (1978). "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems", *Communications of the ACM*, 2: 120–126.

[253] Rosser, J. and J. Schoenfield (1962). "Approximate Formulas for Some Functions of Prime Numbers", *Illinois Journal of Mathematics*, 6: 64–94.

[254] RSA Security Inc. (2008), "Public-Key Cryptography Standards" [online document]. Available at http://www.rsa.com/rsalabs/node.asp?id=2124 (October 2008).

[255] Sakurai, J. J. (1994). *Modern Quantum Mechanics*. Revised by San-Fu Tuan, Reading, Massachusetts: Addison-Wesley.

[256] Satoh, T. (2000). "The Canonical Lift of an Ordinary Elliptic Curve over a Finite Field and Its Point Counting", *Journal of Ramanujan Mathematical Society*, 15: 247–270.

[257] Satoh, T. and K. Araki (1998). "Fermat Quotients and the Polynomial Time Discrete Log Algorithm for Anomalous Elliptic Curves", *Commentarii Mathematici Universitatis Sancti Pauli*, 47: 81–92.

[258] Schiff, L. I. (1968). *Quantum Mechanics*, 3rd ed. New York: McGraw-Hill.

[259] Schindler, W., F. Koeune and J.-J. Quisquater (2001). "Unleashing the Full Power of Timing Attack". Technical report CG-2001/3, Université Catholique de Louvain, Belgium. Available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.23.6622.

[260] Schirokauer, O. (1993). "Discrete Logarithms and Local Units", *Philosophical Transactions of the Royal Society of London, Series A*, 345: 409–423.

[261] Schirokauer, O., D. Weber, and T. Denny (1996). "Discrete Logarithms: The Effectiveness of the Index Calculus Method", *Algorithmic Number Theory—ANTS-II*, *Lecture Notes in Computer Science*, 1122. pp. 337–361. Berlin/Heidelberg: Springer.

[262] * Schneier, B. (2006). *Applied Cryptography*, 2nd ed. New York: John Wiley & Sons.

[263] Schnorr, C. P. (1991). "Efficient Signature Generation for Smart Cards", *Journal of Cryptology*, 4: 161–174.

[264] Schoof, R. (1995). "Counting Points on Elliptic Curves over Finite Fields", *Journal de Théorie des Nombres de Bourdeaux*, 7: 219-254.

[265] Semaev, I. A. (1998). "Evaluation of Discrete Logarithms on Some Elliptic Curves", *Mathematics of Computation*, 67: 353–356.

[266] Shamir, A. (1984). "A Polynomial-Time Algorithm for Breaking the Basic Merkle-Hellman Cryptosystem", *IEEE Transactions on Information Theory*, 30: 699–704.

[267] ——— (1984). "Identity-Based Cryptosystems and Signature Schemes", *Advances in Cryptology—CRYPTO '84*, *Lecture Notes in Computer Science*, 196. pp. 47–53. Berlin/Heidelberg: Springer.

[268] ——— (1997). "How to Check Modular Exponentiation", presented at the rump session of *Advances in Cryptology—EUROCRYPT '97*, May.

[269] ——— (1999). "Factoring Large Numbers with the TWINKLE Device", *Cryptographic Hardware and Embedded Systems—CHES '99*, *Lecture Notes in Computer Science*, 1717. pp. 2–12. Berlin/Heidelberg: Springer.

[270] Shamir, A. and E. Tromer (2003). "Factoring Large Numbers with the TWIRL Device", *Advances in Cryptology—CRYPTO 2003*, *Lecture Notes in Computer Science*, 2729. pp. 1–26. Berlin/Heidelberg: Springer.

[271] Shor, P. W. (1997). "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer", *SIAM Journal of Computing*, 26: 1484–1509.

[272] Shoup, V. (1990). "On the Deterministic Complexity of Factoring Polynomials over Finite Fields", *Information Processing Letters*, 33: 261–267.

[273] Shparlinski, I. E. (1991). "On Some Problems in the Theory of Finite Fields", *Russian Mathematical Surveys*, 46 (1): 199–240.

[274] ——— (1992). *Computational and Algorithmic Problems in Finite Fields*, *Mathematics and its Applications*, 88. Kluwer Academic Publishers.

[275] * Silverman, J. H. (1986). *The Arithmetic of Elliptic Curves*. *Graduate Texts in Mathematics*, 106. Berlin and New York: Springer.

[276] ——— (1994). *Advanced Topics in the Arithmetic of Elliptic Curves*. *Graduate Texts in Mathematics*, 151. New York: Springer.

[277] ——— (2000). "The Xedni Calculus and the Elliptic Curve Discrete Logarithm Problem", *Design, Codes and Cryptography*, 20: 5–40.

[278] Silverman, J. H. and J. Suzuki (1998). "Elliptic Curve Discrete Logarithms and the Index Calculus", *Advances in Cryptology—ASIACRYPT '98*, *Lecture Notes in Computer Science*, 1514. pp. 110–125. Berlin/Heidelberg: Springer.

[279] Silverman, R. D. (1987). “The Multiple Polynomial Quadratic Sieve”, *Mathematics of Computation*, 48: 329–339.

[280] * Sipser, M. (1997). *Introduction to the Theory of Computation*, 2nd ed. Boston: PWS Publishing Company.

[281] B. Skjernaa (2003). “Satoh’s Algorithm in Characteristic 2”, *Mathematics of Computation*, 72: 477–487.

[282] Smart, N. P. (1999). “The Discrete Logarithm Problem on Elliptic Curves of Trace One”, *Journal of Cryptology*, 12: 193–196.

[283] ——— (2002). *Cryptography: An Introduction*. New York: McGraw-Hill. The 2nd edition of this book is available online at http://www.cs.bris.ac.uk/∼nigel/Crypto_Book/ (October 2008).

[284] Smith, P. J. (1993). “LUC Public-Key Encryption: A Secure Alternative to RSA”, *Dr. Dobb’s Journal*, 18 (1): 44–49.

[285] Smith, P. J. and M. J. J. Lennon (1993). “LUC: A New Public Key System”, *IFIP Transactions*, A 37. pp. 103–117. Proceedings of the IFIP TC11, 9th International Conference on Information Security. Computer Security. Amsterdam: North-Holland Co.

[286] Smith, P. J. and C. Skinner (1995). “A Public-Key Cryptosystem and Digital Signature System Based on the Lucas Function Analogue to Discrete Logarithms”, *Advances in Cryptology—ASIACRYPT ’94*, *Lecture Notes in Computer Science*, 917. pp. 357–364. Berlin/Heidelberg: Springer.

[287] Solovay, R. and V. Strassen (1977). “A Fast Monte Carlo Test for Primality”, *SIAM Journal of Computing*, 6: 84–86.

[288] * Stallings, W. (2006). *Cryptography and Network Security*, 4th ed. Upper Saddle River, New Jersey: Prentice-Hall.

[289] Stam, M. and A. K. Lenstra (2001). “Speeding up XTR”, *Advances in Cryptology—ASIACRYPT 2001*, *Lecture Notes in Computer Science*, 2248. pp. 125–143. Berlin/Heidelberg: Springer.

[290] Stein, A. and E. Teske (2005). “Optimized Baby Step-Giant Step Methods”, *Journal of Ramanujan Mathematical Society*, 20 (1): 27–58.

[291] * Stinson, D. (2005). *Cryptography: Theory and Practice*, 3rd ed. Boca Raton, Florida: CRC Press.

[292] Strassen, V. (1969). “Gaussian Elimination Is not Optimal”, *Numerische Mathematik*, 13: 354–356.

[293] Stucki, D., N. Gisin, O. Guinnard, G. Ribordy and H. Zbinden (2002). “Quantum Key Distribution over 67 km with a Plug & Play System”, *New Journal of Physics*, 4: 41.1–41.8.

[294] Sun, H.-M., W.-C. Yang and C.-S. Laih (1999). “On the Design of RSA with Short Secret Exponent”, *Advances in Cryptology—ASIACRYPT ’99*, *Lecture Notes in Computer Science*, 1716. pp. 150–164. Berlin/Heidelberg: Springer.

[295] Swade, D. (2000). *The Cogwheel Brain: Charles Babbage and the Quest to Build the First Computer*. London: Little, Brown and Company.

[296] Trappe, W. and L. C. Washington (2006). *Introduction to Cryptography with Coding Theory*, 2nd ed. Upper Saddle River: Prentice-Hall.

[297] Verheul, E. R. (2001). “Evidence that XTR is More Secure than Supersingular Elliptic Curve Cryptosystems”, *Advances in Cryptology—EUROCRYPT 2001*, *Lecture Notes in Computer Science*, 2045. pp. 195–210. Berlin/Heidelberg: Springer.

[298] Washington, L. C. (2003). *Elliptic Curves: Number Theory and Cryptography*. Boca Raton, Florida: Chapman & Hall/CRC.

[299] Weber, D. (1996). “Computing Discrete Logarithms with the General Number Field Sieve”, *Algorithmic Number Theory—ANTS-II*, *Lecture Notes in Computer Science*, 1122. pp. 337–361. Berlin/Heidelberg: Springer.

[300] ——— (1998). “Computing Discrete Logarithms with Quadratic Number Rings”, *Advances in Cryptology—EUROCRYPT ’98*, *Lecture Notes in Computer Science*, 1403. pp. 171–183. Berlin/Heidelberg: Springer.

[301] Weber, D. and T. Denny (1998). “The Solution of McCurley’s Discrete Log Challenge”, *Advances in Cryptology—CRYPTO ’98*, *Lecture Notes in Computer Science*, 1462. pp. 458–471. Berlin/Heidelberg: Springer.

[302] Western, A. E. and J. C. P. Miller (1968). “Tables of Indices and Primitive Roots”, *Royal Mathematical Tables*, 9, Cambridge: Cambridge University Press.

[303] Wiedemann, D. H. (1986). “Solving Sparse Linear Equations over Finite Fields”, *IEEE Transactions on Information Theory*, 32: 54–62.

[304] Wiener, M. J. (1990). “Cryptanalysis of Short RSA Secret Exponents”, *IEEE Transactions on Information Theory*, 36: 553–558.

[305] Williams, H. C. (1982). “A $p + 1$ Method for Factoring”, *Mathematics of Computation*, 39 (159): 225–234.

[306] Yang, L. T. and R. P. Brent (2001). “The Parallel Improved Lanczos Method for Integer Factorization over Finite Fields for Public Key Cryptosystems”, pp. 106–114. Proceedings of the ICPP Workshops 2001, Valencia, Spain, 3–7 September.

[307] Young, A. and M. Yung (1996). “The Dark Side of “Black-Box” Cryptography, or: Should We Trust Capstone?”, *Advances in Cryptology—CRYPTO ’96*, *Lecture Notes in Computer Science*, 1109. pp. 89–103. Berlin/Heidelberg: Springer.

[308] ——— (1997a). “Kleptography: Using Cryptography Against Cryptography”, *Advances in Cryptology—EUROCRYPT ’97*, *Lecture Notes in Computer Science*, 1233. pp. 62–74. Berlin/Heidelberg: Springer.

[309] ——— (1997b). “The Prevalence of Kleptographic Attacks on Discrete-Log Based Cryptosystems”, *Advances in Cryptology—CRYPTO ’97*, *Lecture Notes in Computer Science*, 1294. pp. 264–276. Berlin/Heidelberg: Springer.

[310] Zheng, Y. (1997). “Digital Signcryption or How to Achieve Cost(Signature & Encryption) $\ll$ Cost(Signature) + Cost(Encryption)”, *Advances in Cryptology—CRYPTO ’97*, *Lecture Notes in Computer Science*, 1294. pp. 165–179. Berlin/Heidelberg: Springer.

[311] ——— (1998a). “Signcryption and Its Applications in Efficient Public Key Solutions”, *1997 Information Security Workshop ISW ’97*, *Lecture Notes in Computer Science*, 1397. pp. 291–312. Berlin/Heidelberg: Springer.

[312] ——— (1998b). “Shortened Digital Signature, Signcryption, and Compact and Unforgeable Key Agreement Schemes”, contribution to IEEE P1363 Standard for Public Key Cryptography.

[313] Zheng, Y. and H. Imai (1998a). “Efficient Signcryption Schemes on Elliptic Curves”. *Proceedings of the IFIP 14th International Information Security Conference IFIP/SEC ’98*, Vienna, Austria, September 1998. Chapman & Hall.

[314] ——— (1998b). “How to Construct Efficient Signcryption Schemes on Elliptic Curves”, *Information Processing Letters*, 68: 227–233.

[315] Zheng, Y. and T. Matsumoto (1996). “Breaking Smartcard Implementations of ElGamal Signatures and Its Variants”, presented at the rump session of *Advances in Cryptology—ASIACRYPT ’96*. Available at http://www.sis.uncc.edu/∼yzheng/publications/ (October 2008).

[316] * Zuckerman, H. S., H. L. Montgomery, I. M. Niven and A. Niven (1991). *An Introduction to the Theory of Numbers*. New York: John Wiley & Sons.

Books marked by stars have Asian editions (at the time of writing this book).

Index

Wisdom begins at the end.
— Daniel Webster

It's always too early to quit.
— Norman Vincent Peale

Someone might say of me that I have only made a bouquet of other people's flowers here, having supplied nothing of my own but the thread to bind them.
— Michel de Montaigne

ACC, 44, 123
active attack, *see* cryptanalyst
adaptive chosen-ciphertext attack, 13
adaptive chosen-message attack, 13
adaptive chosen-plaintext attack, 12
adaptive chosen-signature attack, 13
ADH algorithm, 292, 308
ADH–Gaudry algorithm, 292, 308
Adleman-DeMarrais-Huang algorithm, *see* ADH algorithm
Adleman-Huang primality test, 198
Adleman-Pomerance-Rumely primality test, 198
admissible change of variables, 99, 100
Advanced Encryption Standard, *see* Rijndael
adversary, *see* cryptanalyst
AES, *see* Rijndael
affine
  coordinate, 90
  coordinate ring of a curve, 93
  plane, 90
  space, 90
AKS primality test, 198
algebra, 71
  finitely generated algebra, 71
  polynomial algebra, 71
  quotient algebra, 71
algebraic
  closure, 62, 77, 143, 153
    of a finite field, 64
  dependence, 72
  element, 61, 74, 122
  field extension, 61, 75, 125
  integer, 31, 64, 123, 126
  number, 64
algebraic attack, 478, 490
algebraic geometry, 90, 171
algebraic number theory, 52, 143, 171
algorithm
  constant-time algorithm, 176
  cubic-time algorithm, 176
  deterministic algorithm, 177
  exponential algorithm, 10, 176
  linear-time algorithm, 176
  parallel algorithm, 253
  polynomial-time algorithm, 176
  private algorithm, 13
  probabilistic algorithm, *see* randomized algorithm
  public algorithm, 13
  quadratic-time algorithm, 176
  randomized polynomial-time algorithm, 178, 212, 216
  randomized algorithm, 177, 288
  subexponential algorithm, 10, 176, 178, 218, 264
analytic number theory, 52, 172
annihilator, 70, 73
anomalous attack, 223, 281, 286, 308
anomalous elliptic curve, 107, 110, 223, 286
anonymity, 355
API, 393
arithmetic mean, *see* expectation
ascending chain condition, *see* ACC
associate element, 39, 141
asymmetric-key cryptography, 4

attack, *see* cryptanalysis
attacker, *see* cryptanalyst
authentication, 6, 366
  challenge–response authentication, 6, 368, 375
  mutual authentication, 370
  strong authentication, *see* challenge–response authentication
  unidirectional authentication, 370
  unilateral authentication, 370
  weak authentication, 6, 368
automorphism, 25, 38, 68, 70, 78
average, *see* expectation
axiom of choice, 20

baby-step–giant-step method, 227, 265, 307, 332
Bachet equation, 142
backdoor attack, 406, 424
base station, 498
basis
  complementary basis, 89
  composite basis, 384
  dual basis, 89
  integral basis, 130, 136
    power integral basis, 130
  of a lattice, 301
  of a module, 73
  of a number field, 129
  of a vector space, 66, 72, 443
  ordered basis, 384
  orthonormal basis, 439, 446
  polynomial basis, 85, 204, 215, 273, 384
  reduced basis in a lattice, 302
Bayes rule, 158
Bellcore attacks, 435
Berlekamp's Q-matrix factorization, 217, 234
Bernoulli distribution, 163, 509
Bernstein's factorization circuit, 307
Bézout relation, 35, 42, 45, 58, 275
bijection, *see* bijective function
binary search, 265, 279
binomial distribution, *see* Bernoulli distribution
binomial theorem, 42, 525
birthday paradox, 167, 227, 265, 453, 529
bit
  classical bit, *see* cbit
  leading bit, 383
  least significant bit, *see* trailing bit
  most significant bit, *see* leading bit
  quantum bit, *see* qubit
  string, 3, 383, 386
  trailing bit, 383
block cipher, 466, 494
Blowfish, 4, 467
Blum–Blum–Shub pseudorandom bit generator, 229, 240
Blum–Goldwasser
  decryption, 318
  encryption, 317, 378
  key pair, 317
bootstrapping, 500
Born rule, 443, 444, 453
braid conjugacy problem, 379

cancellation law, 23, 135
canonical lift of an elliptic curve, 222
cardinality
  of a basis of a vector space, 67
  of a finite field, 80
  of a set, 17, 20
Carmichael number, 196, 202
carry, 182
Cauchy sequence, 91, 143, 147, 153
  convergence of Cauchy sequence, 143
Cauchy's theorem, 28
CBC mode, 481
CBCMAC, 9
cbit, 440
ceiling, 50
central limit theorem, 165, 414
centralizer, 30
centre of a group, 30
certificate
  public-key certificate, 9, 337, 393
  revocation list, 9
certification authority, 9
CFB mode, 481, 487
CFRAC, 307
chain, 19
  addition chain, 195, 285
  ascending chain, 41, 44, 123
  descending chain, 152
  stationary chain, 44, 123
challenge, 368, 371
challenge–response authentication, *see* authentication
characteristic, 34, 43, 75, 79, 80, 99, 111, 145, 204

Chaum's RSA blind signature, 355, 378
Chaum–Van Antwerpen undeniable signature, 358, 378
  denial protocol, 359
  generation, 358
  key pair, 358
  verification, 359
Chinese remainder theorem, *see* CRT
Chor–Rivest
  decryption, 322
  encryption, 321, 333, 378
  key pair, 321
chord and tangent rule, 102
chosen message attack, 435
chosen-ciphertext attack, 12, 395
chosen-message attack, 13
chosen-plaintext attack, 12
chosen-signature attack, 13
ciphertext, *see* message
ciphertext-only attack, 12
circle, 90
claimant, 6, 366, 370
class equation, 28, 30
closest pairwise scheme, 513
closest vector problem, *see* CVP
coding theory, 234
collision, 8, 491
collision attack, 478
collision resistance, 8, 491, 493, 496
commitment, 371
commutative algebra, 120, 171
complete field, 147
complete residue system, 56
complete space, 439
completion, 91, 143, 147, 148
complex conjugate, 25, 38, 79, 127, 439
complex embedding, 126, 258
complex number, 17, 32, 143
complexity classes, 518
complexity theory, 234, 518
congruence, 144
  linear congruence, 48, 249, 263, 267, 270
  modulo an ideal, 37
  of integers, 46
  of polynomials, 58
  polynomial congruence, 203
  quadratic congruence, 48, 57, 248, 259, 262
conjugacy class, 30
conjugate, 77, 84, 126, 131, 324
  in a group, 30
  of a polynomial function, 108, 113
conjugate gradient method, 297
contaminated cryptosystem, 424
content of a polynomial, 63
continued fraction, 455
continued fraction method, *see* CFRAC
controlled NOT, 447
convergence
  of a sequence, 147
  of a series, 153
  of a sum, 147
Coppersmith and Winograd's algorithm, 294
Coppersmith's algorithm, 276, 308
coprime, 42, 44
correlation coefficient, 161, 164, 165
correspondence theorem, 30
coset, 23, 36, 46, 116
  left coset, 23
  right coset, 23
covariance, 161, 164, 166
critical line, 55
critical strip, 55
CRL, *see* certificate revocation list
CRT, 39, 46, 203, 219, 240, 290, 295, 313, 394, 417, 422, 435, 526
crypt password, 368
cryptanalysis, 2, 11, 154, 406
  differential cryptanalysis, 471, 478
  linear cryptanalysis, 471, 478
  quantum, 452
cryptanalyst, 3, 5, 11, 310
  active cryptanalyst, 12, 335, 336, 501
  passive cryptanalyst, 12, 334
    models of passive cryptanalyst, 12
cryptography, 2
  black-box cryptography, 432
  public-key, 178
  quantum cryptography, 448
cryptology, 2
CSM, *see* cubic sieve method
cubic sieve method, 263, 272, 280, 307, 308
curve
  affine curve, 90, 97
  isomorphic curve, 95, 99, 101
  projective curve, 92, 97

real curve, 90
smooth curve, 93, 96, 98, 101, 107, 111
CVP, 242
cyclotomic polynomial, 292

deciphering, *see* decryption
decoding, 3, 382, 397, 399
decryption, 3
function, 3, 5
Dedekind domain, 125, 131, 141
Dedekind's theorem on ramification, 137
degree
of a divisor, 96
of a field extension, 74, 125
of a monomial, 91
of a number field, 125, 258
of a polynomial, 32, 63, 92
of a polynomial function, 108, 114
dehomogenization of a polynomial, 92, 97
denial protocol of an undeniable signature, 358
dense set, 153
density
of a knapsack set, 300
derivative
formal derivative, 64, 129, 149, 213
partial derivative, 93
DES, 4, 6, 368, 414, 435, 467, 483, 496
decryption, 471
encryption, 468
key schedule, 468
test vectors, 471
DES challenge, 471
determinant
of a lattice, 304
Deutsch algorithm, 445, 461
DHP, 10, 11, 179, 241, 319, 321, 323, 334
dictionary-based attack, 367
Diffie–Hellman key exchange, 334, 378, 393, 402, 431
Diffie–Hellman problem, *see* DHP
Digital Signature Algorithm, *see* DSA
Digital Signature Standard, *see* DSA
dimension
of a vector space, 67
Dirac delta, 422
Dirac's ket notation, 438
direct product
of groups, 29, 30
of modules, 69
of rings, 32
direct sum
of groups, 107
of modules, 69
of vector spaces, 72
Dirichlet's theorem on primes in arithmetic progression, 55
Dirichlet's unit theorem, 139, 260
disavowal protocol, *see* denial protocol
discrete logarithm, 240, 279
discrete logarithm problem, *see* DLP
discrete valuation, 141, 286
discrete valuation domain, *see* DVR
discrete valuation ring, *see* DVR
discriminant
of a number field, 130, 136
of a polynomial, 64, 128
of algebraic elements, 128
of an elliptic curve, 100
distance
Archimedean distance, 143
p-adic distance, 143, 146
distinct-degree factorization, 212
divisibility
in a ring, 39
of ideals, 135
of integers, 44
division polynomial, 105, 109, 220
Cantor's division polynomial, 228
divisor, 96, 115, 282, 284
class group, *see* Picard group
equivalent divisor, 116
greatest common divisor, 116
group, 96
of a rational function, 96, 115, 282
prime divisor, 293
principal divisor, 96, 102, 115, 285
reduced divisor, 117, 224, 285, 293
sum of reduced divisors, 225
semi-reduced divisor, 116, 225
smooth divisor, 293
DLP, 9, 11, 179, 204, 210, 240, 241, 243, 264, 281, 307, 321, 323, 334, 343, 346, 356, 357, 374, 379, 382, 438, 456, 520
generalized DLP, 180, 240
domain, *see* integral domain

factorial domain, *see* UFD
dot product, 438
double-and-add algorithm, 218, 225
DPA, *see* differential power analysis
DSA, 346, 378, 418
key pair, 347
prime, 347
signature
generation, 347
verification, 347
dual space, 72
DVD, *see* DVR
DVR, 141, 152, 221, 286

early abort strategy, 269, 275, 278
eavesdropper, *see* cryptanalyst
ECB mode, 480
ECDHP, 241
ECDLP, 10, 107, 235, 241, 281, 308, 382
ECDSA, 348, 365, 378
key pair, 348
signature
generation, 349
verification, 349
ECM, *see* elliptic curve method
E-D-E mode, 480
Eisenstein's criterion, 63
elementary number theory, 171
ElGamal
decryption, 320
encryption, 11, 319, 333, 378, 431
key pair, 319, 343
signature, 343, 364, 378, 419, 424, 430
generation, 343
verification, 344
ellipse, 90, 98
elliptic curve, 27, 98, 119, 171, 218, 234, 241, 255, 323, 348, 378, 385, 393, 436
discrete logarithm problem, *see* ECDLP
group of rational points on an elliptic curve, 101
isogenous elliptic curve, 222
subfield elliptic curve, 224
with complex multiplication, 222, 224
Elliptic Curve Digital Signature Algorithm, *see* ECDSA
elliptic curve method, 255, 307
second stage in elliptic curve method, 257
elliptic curve point, 385, 389
compressed representation, 385, 389
hybrid representation, 385, 389
lossy compressed representation, 385, 390
LSB compressed form, 385, 390
SORT compressed form, 385, 389
uncompressed representation, 385, 389
enciphering, *see* encryption
encoding, 3, 382, 397, 398, 402
encryption, 3, 310, 368
deterministic encryption, 315
double encryption, 478
function, 3, 5
multiple encryption, 478, 486
plaintext-aware encryption, 395
probabilistic encryption, 315
triple encryption, 478
encryption–decryption protocol, 3
endomorphism, 25, 38, 68, 70
endomorphism ring of an elliptic curve, 222
enemy, *see* cryptanalyst
entanglement, 441, 446
entity, 3
EPOC encryption, 379
equal-degree factorization, 212
equivalence class, 18, 20
equivalence relation, *see* relation
ERH, 55, 198
error function, 163
error-correcting code, 379
ESign signature, 379
Euclidean division, 45, 58
Euclidean domain, 43, 140
Euler constant, 262
Euler zeta function, 54
Euler's criterion, 49, 57, 196, 524
Euler's theorem, 47
Euler's totient function, 47, 48, 57, 180, 239
existential forgery, 355, 364, 492
expectation, 159, 162, 163, 164, 166
exponent of a group, 227, 228
Extended Riemann hypothesis, *see* ERH
external direct product
of groups, 30

factor base, 248, 263, 267, 274, 275, 276, 280, 293
fault
induced fault, 416

latent fault, 416, 422
transient fault, 416
fault analysis, 416, 423, 435
differential fault analysis, 436
FEAL, 4, 467
Feige–Fiat–Shamir (FFS) zero-knowledge protocol, 372, 378, 420
Feistel cipher, 468
Fermat number, 56
Fermat prime, 56, 243
Fermat's little theorem, 47, 81, 195, 524
Fermat's primality test, 202
FFT, *see* Fast Fourier Transform
field, 33, 37
algebraically closed field, 62, 92, 143
fixed field, 78
intermediate field, 81
of fractions, *see* quotient field
perfect field, 61, 79, 81, 126
quotient field, 44, 93, 121, 123, 125, 145
residue field of a local ring, 140, 152
subfield, 34
field extension, 34, 58, 60, 65, 74, 88, 323
finite field extension, 74, 125
infinite field extension, 74, 143
representation of field extension, 59
finite field, 27, 33, 80, 106, 171, 204, 234, 241, 264, 389, 472
of characteristic 2, 204, 384
prime finite field, 204, 384
representation of finite fields, 85
uniqueness of a finite field, 81
finite point on a curve, 93, 97, 113
finite valuation, 146
FIPS, 404
first element, 19
first pre-image resistance, 8, 492, 493, 496
floor, 50
formal sum, 95
Fourier transform, 454, 457
fast Fourier transform, 181, 185
quantum Fourier transform, 455, 459
Frobenius map, 89, 220, 221, 226, 525
function, 18
analytic function, 53
bijective function, 19, 22, 154
composition of functions, 19
compression function, 492, 496
constant function, 18
hash function, *see* hash
holomorphic function, 53
identity function, 18
injective function, 18
inverse function, 20, 22
non-constant function, 18
one-one function, 18
one-way function, 520
trapdoor one-way function, 521
onto function, 19
periodic function, 453
surjective function, 19, 25
function field, 93, 94
fundamental theorem of algebra, 63, 127, 524
fundamental theorem of Galois theory, 78, 525
fundamental unit, 139, 272

Galois extension, 78, 80, 89
gap failure in NTRU decryption, 331
Gauss' law of quadratic reciprocity, 50
Gauss' lemma, 50
Gauss' Li function, 53
Gaussian distribution, *see* normal distribution
Gaussian elimination, 249, 263
structured Gaussian elimination, 296
Gaussian integer, 139, 141
Gaussian integer method, 271, 308
gcd, 220
binary gcd, 187
Euclidean gcd, 45, 58, 187, 194, 207
extended gcd, 10, 189, 190, 194, 207, 275
in a UFD, 41
monic gcd, 58
of ideals, 142
of integers, 44, 187
Geffe generator, 489, 491
general linear group, 22
generator
of a group, 26, 57, 211, 240
of a vector space, 65
of an algebra, 71
of an ideal, 35, 259
genus, 111, 118, 224, 292
geometric distribution, 166

GKR undeniable signature, *see* RSA-based undeniable signature
Goldreich–Goldwasser–Halevi encryption, 379
Goldwasser–Micali
  decryption, 316
  encryption, 315, 378
  key pair, 315
Gordon's algorithm for generating strong primes, 199, 311
Gram–Schmidt orthogonalization, 302, 304
gray code, 279
greatest common divisor, *see* gcd
group, 21
  Abelian group, 22, 65, 69
    type of an Abelian group, 107
  additive group, 22
  commutative group, 22
  cyclic group, 26, 30, 47, 57, 82, 87, 89, 107, 110, 210, 240, 265, 489
  finite group, 22
  finitely generated group, 26
  free Abelian group, 96
  Galois group, 78
  multiplicative group, 22
  of automorphisms, 25, 89
  quotient group, 24, 25
  subgroup, 23
    normal subgroup, 24, 29
    Sylow subgroup, 28, 201
    trivial subgroup, 23
Guillou–Quisquater (GQ) zero-knowledge protocol, 373, 379

Hadamard inequality, 304
Hadamard transform, 443, 446, 447, 448, 453
Hamming weight, 414
harmonic series, 250, 262
hash, 6, 8, 340, 352, 369, 395, 398, 431, 491
  collision-resistant hash function, 492
  cryptographic hash function, 492, 495
  keyed hash, 9, 492
  one-way hash function, 492
  unkeyed hash, 9, 492
Hasse's theorem, 106, 219, 256
Hasse–Weil interval, 226, 294
Hausdorff's maximal principle, 20
HECDHP, 241
HECDLP, 10, 241, 292, 308
Hensel lifting, 150, 203, 240, 250, 295, 330
Hensel's lemma, 149
hidden field equation, 379
hidden monomial cryptosystem, 380
Hilbert space, 439, 446
Hilbert's basis theorem, 36, 124, 131
HMAC, 9
homogeneous
  coordinate, *see* projective coordinate
homogenization of a polynomial, 92, 97, 113
homomorphism, 25, 37, 43, 68, 70, 71, 89
  of finite type, 71
hyperbola, 90
hyperelliptic curve, 27, 96, 98, 111, 171, 224, 235, 241
  discrete logarithm problem, *see* HECDLP

ICM, *see* index calculus method
IDEA, 4, 467
ideal, 34, 69
  coprime ideal, 38, 120
  finitely generated ideal, 35, 123
  fractional ideal, 132
    inverse of a fractional ideal, 133, 134
    principal fractional ideal, 132
    product of fractional ideals, 132
  homogeneous ideal, 98
  integral ideal, 132
  intersection of ideals, 35, 120, 524
  maximal ideal, 36, 42, 120, 125, 140, 147
  prime ideal, 36, 120, 125, 140
  principal ideal, 35, 142, 152
  product of ideals, 120, 132
  radical ideal, 44
  relatively prime ideal, *see* coprime ideal
  root ideal, 44
  sum of ideals, 35, 120
  unit ideal, 35
  zero ideal, 35
idempotent element, 33
identification, 366
identity element, 21
identity-based
  encryption, 378, 382
  signcryption, 378
IEEE P1363 standard, 382, 404
IETF, 404

IFP, 9, 10, 180, 195, 239, 243, 307, 310, 355, 379, 382, 438, 444, 455, 520
  generalized IFP, 239
  RSA IFP, 239
image, 19, 25, 38, 68, 70, 72
impersonation attack, 366
index
  of a group element, *see* discrete logarithm
  of a subgroup, 24
index calculus method, 264, 267, 292, 308
  basic index calculus method, 268, 274, 280
  first stage in index calculus method, 267, 268, 270, 274, 275, 276, 294
  precomputations in index calculus method, 268
  second stage in index calculus method, 267, 268, 271, 274, 276, 277, 280
inertial degree, 137, 260
infinite valuation, 146
initialization vector, 481
injection, *see* injective function
inner product, 301, 438, 442, 446
inner product space, 439
input size, 176
integer, 16, 22, 32, 44, 143, 384, 388
  addition of integers, 182
  decimal expansion of integers, 56
  Euclidean division of integers, 185
  left-shift of integers, 187
  modular addition of integers, 22
  modular inverse of integers, 23, 46
  modular multiplication of integers, 23
  multiple-precision integer, 180, 234
    representation of multiple-precision integer, 181, 194
  multiplication of integers, 183
  power-smooth integer, 246, 247
  rational integer, 125
  smooth integer, 54, 244, 248, 256, 258, 261, 268, 270, 272
    density of smooth integers, 54, 244
  square root, 204, 248
  squaring of integers, 184
  subtraction of integers, 182
integer factorization problem, *see* IFP
integral
  closure, 123, 140
  dependence, 122
  element, 122
  extension, 123, 139
integral domain, 33, 37, 43, 63, 121, 145
integrally closed, 123, 125
internal direct product, *see* direct product
interval
  closed interval, 154
  open interval, 154
intruder, *see* cryptanalyst
inverse, 22
irreducibility testing, 83
  of polynomials, 209
irreducible element, 39
isogeny, 222
  non-constant isogeny, 222
  zero isogeny, 222
isomorphism, 25, 37, 68, 70, 217
isomorphism theorem, 26, 29, 38, 43, 68, 70, 72
iteration count, 367

j-invariant, 100, 107, 108, 111, 222, 525
Jacobi symbol, 51, 200, 203, 315
Jacobian, 96, 102, 111, 116, 118, 224, 241, 292

Kaltofen–Shoup polynomial factorization algorithm, 214
Karatsuba multiplication, 185
Kerberos, 436
kernel, 25, 38, 68, 70, 72, 222
key, 3, 424, 466, 486
  anti-palindromic key, 483
  decryption key, 3
  encryption key, 3
  establishment, 500
  pair, 4, 5, 10, 310, 368, 370
    session pair, 320, 328, 333, 343, 344, 345, 347, 349, 356, 430, 431
  path key establishment, 500
  predistribution, 500
    location-aware predistribution, 513
    matrix-based predistribution, 511
    pairwise predistribution, 506, 513
    polynomial-pool-based predistribution, 508, 515
    random predistribution, 502
  private key, 4, 310, 407
  public key, 4, 310
  ring, 500
  secret key, 4

symmetric key, 4
weak key, 483
key agreement, *see* key exchange
key exchange, 5, 334
quantum key exchange, 438, 448, 451, 461
key-only attack, 13
KLCHKP encryption, 379
kleptography, 425
knapsack, 300, 304, 321
superincreasing knapsack, 300
knapsack problem, *see* SSP
known-ciphertext attack, 12
known-message attack, 13, 435
known-pair attack, 12, 13
known-plaintext attack, 12, 479, 489
known-signature attack, 13
Kronecker delta, 89, 122

L^3 algorithm, *see* Lenstra–Lenstra–Lovasz algorithm
Lagrange's interpolation formula, 63, 508, 527
Lagrange's theorem, 24
Lanczos method, 298
large prime variation, 251, 269, 271, 275, 278
largest element, 19
Las Vegas algorithm, 178, 180, 208, 212
last element, 19
lattice, 242, 301, 332, 353
basis reduction in a lattice, 300, 302, 304, 308, 332, 379
dimension of a lattice, 301
lcm, 41
of ideals, 142
of integers, 44
leading coefficient, 32, 114
least common multiple, *see* lcm
least element, 19
Legendre form, 111
Legendre symbol, 49, 138, 200, 218, 242
length
of a vector, 301
Lenstra–Lenstra–Lovasz algorithm, 239, 300, 308
LFSR, 487, 490
maximum-length LFSR, 489, 490
lies over, 131
lifting, *see* Hensel lifting
limit, 147
line at infinity, 91
linear algebra, 171
linear complexity, 491
linear dependence, 66
linear feedback shift register, *see* LFSR
linear order, *see* total order
linear sieve method, 270, 275, 280, 308
linear transformation, *see* vector space homomorphism
localization, 121, 124, 140
at a prime ideal, 121
lower bound, 19
LSM, *see* linear sieve method
LUC encryption, 379
Lucas sequence, 379

MAC, 492
map, *see* function
mask-generation function, 396, 399
matrix, 22
conjugate transpose of a matrix, 442
invertible matrix, 22
symmetric matrix, 296
transformation matrix, 73
unitary matrix, 442, 446
maximal element, 19, 124
maximal spectrum, 120
maximum element, 19
McEliece encryption, 379
MD2, 9, 401
MD4, 494
MD5, 9, 401, 494
MDC, 492
measurement, 443
in the classical basis, 444
meet-in-the-middle attack, 479, 486
memoryless random variable, 167
Menezes–Qu–Vanstone key exchange, 338, 378
Merkle's meta method, 492
Merkle–Hellman cryptosystem, 308
Mersenne number, 56, 210
Mersenne prime, 56
message
ciphertext message, 3, 10, 310
expansion, 320
plaintext message, 3, 10, 310
unconcealed message, 333
Message digest, *see* hash
metric, 145, 146

equivalent metric, 146
space, 146
Miller–Rabin primality test, 177, 196, 244
minimal element, 19
minimal polynomial, 131
minimum element, 19
Möbius function, 83
Möbius inversion formula, 83, 292
modular
addition, 190, 208
exponentiation, 11, 190, 197, 208, 269, 407, 408
inverse, 190, 208
multiplication, 190, 208
square root, 202, 219, 231, 234, 239
subtraction, 190
modular polynomial, 222
module, 69, 122
cyclic module, 70
finitely generated module, 70, 122, 132
free module, 70, 73, 129
quotient module, 70
submodule, 69
monomial, 91
Monte Carlo algorithm, 177, 196
Montgomery exponentiation, 192, 193
Montgomery multiplication, 192, 234
Mordell–Weil theorem, 289, 527
morphism between curves, 95, 222
MOV reduction, 223, 281, 282, 308, 385
multi-path key reinforcement, 505
multiplicative(ly closed) set, 121, 140
multiplicity, 40, 61, 95, 239

natural number, 16
neighbour, 500
NFSM, *see* number field sieve method
nilpotent element, 33, 42
nilradical, 42, 44
no-cloning theorem, 443, 450, 453
node capture, 499, 501, 503, 505, 509, 510, 511
non-linear combination generator, 489
non-linear filter generator, 489
non-singular point on a curve, *see* smooth point
non-supersingular elliptic curve, 100, 103, 107, 109, 219, 281, 385
norm
absolute norm, 85
Archimedean norm, 146, 147
centered norm, 352
equivalent norm, 146
in an inner product space, 439
non-Archimedean norm, 146
of a divisor, 117, 119
of a polynomial function, 108, 113
of an algebraic number, 128, 136, 260
of an ideal, 136, 142, 259
multiplicativity of of an ideal, 136
on a field, 146
over finite field, 85, 89
transitivity of over finite field, 89
p-adic norm, 146, 152
trivial norm, 146
normal basis, 86, 325, 384
optimal normal basis, 87, 234
primitive normal basis, 86
normal distribution, 163
standard normal distribution, 163
normal domain, 123, 125, 140
normal element, 86
normal extension, 78, 80, 81
NTRU
decryption, 331
encryption, 328, 334, 378, 386
key pair, 328
signature, *see* NTRUSign
NTRUSign signature, 352, 378
generation, 353
key pair, 352
verification, 354
nullity, 69
number field, 125, 171, 258, 289, 379
norm-Euclidean number field, 142
quadratic number field, 127, 131, 138, 139, 141
ring of integers of a number field, 125, 258
signature of a number field, 127, 139
totally complex number field, 127
totally real number field, 127
number field sieve method, 258, 272, 307, 308
general number field sieve method, 258, 273
special number field sieve method, 258, 272
number ring, 31, 125
monogenic number ring, 130, 137
Nyberg–Rueppel
key pair, 345

signature, 345, 378
generation, 345
verification, 346

O-notation, 175, 179
soft O-notation, 176
OAEP, *see* optimal asymmetric encryption procedure
octet, 383
leading octet, 383
least significant octet, 383
most significant octet, 383
string, 383, 386
trailing octet, 383
OFB mode, 482, 487
Okamoto–Schnorr blind signature, 357, 378
one-more forgery, 355
strong one-more forgery, 355
one-time pad, 486
one-way function, 367
operation
associative operation, 21, 31
binary operation, 21, 31
commutative operation, 22, 31
distributive operation, 31
opponent, *see* cryptanalyst
opposite of a point, 102, 112, 119, 218, 385
optimal asymmetric encryption procedure, 393, 395
order
exponential order of growth, 175, 179
modulo an integer, 57
of a group, 27, 240
of a polynomial function at a point on a curve, 114, 119
of a rational function at a point on a curve, 94, 115
of an element in a group, 27, 217, 260
of growth of a function, 175, 179
poly-logarithmic order of growth, 175
polynomial order of growth, 175, 179
subexponential order of growth, 176
ordinary point on a hyperelliptic curve, 112
Ostrowski's theorem, 147

p-adic
expansion, 151, 153, 287
integer, 143, 151, 287
canonical representation of integers, 144
product of integers, 144, 152
sum of integers, 144, 152
number, 143, 152, 221, 287
field of numbers, 145
product of numbers, 152
sum of numbers, 152
projection map, 221
p-coherent sequence, 144
packed representation of finite fields, 217, 385
PAP, *see* pretty awful privacy
parabola, 90
PARI, 182, 234
partial order, 18, 20
partition of a set, 18
party, 3
passive attack, *see* cryptanalyst
password, 6, 366, 382, 393
in UNIX, 6, 367
Pépin's primality test, 203
period, 453
personal identification number, 366
PGP, *see* pretty good privacy
Picard group, 96, 115
PID, 35, 43, 58, 124, 135, 139, 140, 149, 152, 272
pigeon-hole principle, 524
PIN, *see* personal identification number
PKCS, 393, 404
#1, 393
#3, 402
PKI, 9
plaintext, *see* message
Pohlig–Hellman method, 210, 266, 307, 321
point at infinity, 91, 92, 97, 98, 111, 113
point counting
on elliptic curves, 219, 234
on hyperelliptic curves, 225
Poisson distribution, 166
pole, 94, 115, 282
Pollard $p-1$ method, 245, 307
second stage, 246
Pollard rho method, 244, 265, 307
polynomial, 32
addition of polynomials, 206
bivariate polynomial, 90
connection polynomial, 489, 491
Euclidean division of polynomials, 206, 215
factorization of polynomials, 212, 234
Shoup's algorithm, 234

homogeneous polynomial, 92, 94, 97
irreducible polynomial, 58, 89, 126, 153, 178, 208, 214, 258, 272, 273, 489
algorithm for computing irreducible polynomial, 84
minimal polynomial, 61, 75, 79, 126, 128, 299, 324
monic polynomial, 32, 122
multiplication of polynomials, 206
multivariate polynomial, 32, 90, 379
over a UFD, 63
primitive polynomial, 63, 489
smooth polynomial, 273, 275
square-free polynomial, 61, 64, 212
squaring of polynomials, 206
univariate polynomial, 32
polynomial function, 93, 108, 113, 119
poset, *see* partially ordered set
power analysis, 411, 423, 435
differential power analysis, 413
simple power analysis, 411
power residue, 57
power series, 491
power trace, 411, 412, 415
pretty awful privacy, 426, 432
pretty good privacy, 426
primality testing, 195, 234, 519
deterministic primality testing, 197
prime, 44, 234
completely split prime, 137
in a number ring, 137, 259
in a ring, 39
inert prime, 137
infinitude of primes, 44
ramified prime, 137
safe prime, 199, 210, 248, 264, 311, 358
strong prime, 199, 210, 234, 248, 264, 311
totally ramified prime, 137
prime number theorem, 53, 199, 219, 270
prime spectrum, 120
primitive element, 86, 87, 264, 279
in a cyclic group, *see* generator
primitive normal element, 86
primitive part of a polynomial, 63
primitive root, 26, 243
principal ideal domain, *see* PID
probability, 154, 172, 177
conditional probability, 157
cumulative probability, 155
density function, 154
distribution function, 154
joint probability, 155, 165
marginal probability, 166
mass function, 154
problem
intractable problem, 178, 238, 310, 371, 406, 452
NP-complete problem, 300
polynomial-time equivalent problems, 179, 180
randomized polynomial-time equivalent problems, 179, 263
projective
coordinate, 91, 436
line, 91
plane, 91
space, 91
properly complex embedding, 127
prover, 6, 366
pseudoprime, 195
Euler pseudoprime, 196
Solovay–Strassen pseudoprime, *see* Euler pseudoprime
strong pseudoprime, 196
pseudorandom bit generator, 228, 235, 419
cryptographically strong pseudorandom bit generator, 229, 235
seed for a pseudorandom bit generator, 229, 230
PSS, *see* probabilistic signature scheme
public-key cryptography, 2, 4, 9
Public-Key Cryptography Standards, *see* PKCS
public-key infrastructure, *see* PKI
purely inseparable
element, 79
extension, 79

q-composite scheme, 504
QSM, *see* quadratic sieve method
quadratic non-residue, 49, 55
quadratic reciprocity, *see* Gauss' law of quadratic reciprocity
quadratic residue, 49, 231, 242
quadratic residuosity problem, 242, 315
quadratic sieve method, 248, 263, 270, 307
multiple-polynomial quadratic sieve method, 252, 307
quantum computation, 438, 445, 461
quantum error correction, 442
quantum mechanics, 438, 461

Copenhagen interpretation, 461
first axiom, 439
fourth axiom, 443
second axiom, 440
system, 438, 439
third axiom, 442
quantum register, 441, 445, 453, 456, 459
qubit, 440, 447
quotient
of Euclidean division, 45, 58, 185, 206

Rabin
decryption, 314
encryption, 313, 333, 378
key pair, 313, 342
signature, 342, 364, 378, 418
generation, 342
verification, 342
radical of an ideal, 44
radix, 181, 193
ramification, *see* prime
ramification index, 137
random experiment, 154
random variable, 154, 165, 409
continuous random variable, 154
discrete random variable, 154
independent random variables, 155, 157, 160, 161
product of random variables, 156
sum of random variables, 156
rank
of a linear transformation, 69, 295
of a module, 73, 129
of an Abelian group, 107
of an elliptic curve, 289
rational function, 32, 42, 93, 94, 95, 108, 113, 121
rational number, 16, 22, 32, 125, 143
rational point on a curve, 90, 92
ray, 439
RC5, 4
real embedding, 127
real number, 16, 32, 143
extended real number, 91
receiver, 3, 310
reduced residue system, 56
reduction
of a ring, 42
polynomial-time reduction, 178
randomized polynomial-time reduction, 179
redundancy, 365
regular point on a curve, *see* smooth point
relation, 17
anti-symmetric relation, 17
equivalence relation, 17, 20, 46, 91, 95, 100, 121, 439
in a sieve method, 248, 253, 259, 263
in ICM, 267, 270, 272, 274, 275, 276, 280, 294
reflexive relation, 17
symmetric relation, 17
transitive relation, 17
relatively prime, *see* coprime
remainder
of Euclidean division, 45, 58, 185, 206
residue field, 140
residue list sieve method, 271, 308
response, 368, 371
reversible XOR, 442, 447
Riemann hypothesis, 55
Riemann zeta function, 55
Rijndael, 4, 5, 435, 467, 468, 472, 483
decryption, 476
encryption, 474
key schedule, 473
test vectors, 478
ring, 31, 144
commutative ring, 31
extension, 34
factorial ring, *see* UFD
local ring, 140, 152
Noetherian ring, 35, 124, 131, 140
of fractions, *see* localization
polynomial ring, 32, 42, 65, 69
quotient ring, 37, 42, 124
reduced ring, 42
subring, 34
superring, *see* ring extension
with identity, 31
zero ring, 31, 140
ring element, 386, 390
binary ring element, 386, 392
RIPEMD, 494
root, 59, 215
adjoining of root, 60, 75
multiple root, 61, 112
of an ideal, 44
simple root, 61

root of unity, 282, 291
  primitive root of unity, 283
rotational invariance property of a lattice, 354
RSA, 239, 242, 332, 380, 393, 406, 417, 424, 425, 432, 455, 521, 527
  decryption, 10, 312, 395, 397, 400
  encryption, 10, 310, 378, 394, 396, 400
  key pair, 10, 310, 341, 364, 394, 425
  modulus, 311, 333, 425
  multi-prime RSA, 394
  signature, 341, 364, 378, 422
    generation, 341, 398, 401
    verification, 342, 399, 401
RSA key inversion problem, 242, 310
RSA Lab standards, *see* PKCS
RSA problem, 242, 310, 355, 373
RSA-based undeniable signature, 360, 378
  denial protocol, 361
  generation, 360
  key pair, 360
  verification, 361
RSAREF, 407
running time, *also see* algorithm
  average running time, 176, 179
  best running time, 176, 179
  deterministic running time, 177
  expected running time, 176, 178, 179, 245, 521
  subexponential running time, 243, 267, 307
  worst running time, 176, 178, 179, 521
Russell's paradox, 21

SAFER, 4, 467
salt, 367, 398, 528
sample, 164
sample space, 154
Satoh–FGH algorithm, 221, 223, 348
scalar multiplication, 65, 69
scalar product, 438
Schnorr
  key pair, 344, 356
  signature, 344, 378
    generation, 344
    verification, 345
Schnorr blind signature, 356, 378
Schnorr zero-knowledge protocol, 374, 379
Schrödinger cat, 440, 444
SEA algorithm, 219, 223, 227, 348
second pre-image resistance, 8, 491, 493, 496
secret sharing, 8
secret-key cryptography, 4
Secure Hash Algorithm, *see* SHA
security, 11, 238
  provable security, 10, 406, 438
  unconditional security, 486
semi-Witt decomposition, 221
sender, 3, 310, 355
sensor network, 498
  maximum supportable size of a sensor network, 507
  mobile sensor network, 499
separable
  element, 79
  extension, 78, 79
  polynomial, 79
set
  axiomatic definition of a set, 16
  complement of a set, 17
  difference of sets, 17
  disjoint union of sets, 17, 21
  disjoint set, 17
  empty set, 17
  extraordinary set, 21
  finite set, 17, 20
  infinite set, 17
  intersection of sets, 17
  ordinary set, 21
  power set, 20
  product of sets, 17
  subset, 17
    proper subset, 17
  union of sets, 17
SETUP, 425, 436
SHA, 9, 347, 348, 395, 401, 494, 496
shared secret, 5
Shor's algorithm, 453, 459, 461
short vector in a lattice, 301
shortest vector problem, *see* SVP
side-channel attack, 11, 406, 407, 435
sieve, 249, 261, 270, 280
  incomplete sieve, 251, 254, 271
  of Eratosthenes, 264
  polynomial sieve, 278
signature, 5, 9, 13, 340, 369, 375, 417
  blind signature, 355
  forged signature, 6

probabilistic signature, 393, 398
shortened signature, 363
undeniable signature, 357
with appendix, 6, 340
with message recovery, 6, 340, 365
signcryption, 362, 378
signer, 340, 355
simple extension, 75, 82, 126
simple order, *see* total order
sleep deprivation attack, 501
sliding window method of exponentiation, 194
small-subgroup attack, 335
smallest element, 19
smart card, 411, 412, 416, 424, 435
SmartASS method, *see* anomalous attack
smooth point on a curve, 93
SPA, *see* simple power analysis
span, 68
sparse linear system, 269, 272, 274, 276, 277, 294, 308
special linear group, 23
special point on a hyperelliptic curve, 112, 119
splitting field, 60, 64, 76
SQRTP, 201, 230, 240, 263, 313, 317, 372
square attack, 478
square root method, 264
square root problem, *see* SQRTP
square-and-multiply exponentiation, 191, 407, 412, 417
square-free factorization, 212
SSP, 241, 300, 308, 321, 382
low-density SSP, 301
standard deviation, 160, 164
straight line, 90
Strassen's algorithm, 294
stream cipher, 486
asynchronous stream cipher, 487
synchronous stream cipher, 487
structure homomorphism of an algebra, 71
structure theorem
for Jacobian of a hyperelliptic curve, 226
of a finitely generated algebra, 72
of a finitely generated module, 70
of an Abelian group, 107
of an elliptic curve group, 107
subject, 3
subset sum problem, *see* SSP
substitution homomorphism, 38
superposition, 440, 441, 453
supersingular elliptic curve, 100, 103, 107, 110, 223, 281, 284
superuser, 367
support
of a divisor, 96
surjection, *see* surjective function
SVP, 242
Sylow's theorem, 28
symmetric-key cryptography, 4, 466

Teichmüller lift, 221
Teichmüller representative, 153
tensor product, 440
term, 91
timestamp, 370
timing attack, 407, 423, 435
Toffoli gate, 448
Tonelli and Shanks algorithm, 201
torsion element, 73, 139, 289, 292
torsion point on a curve, 105, 106, 220, 227, 282
torsion subgroup, 292
torsion-free module, 73, 139
total order, 18, 20, 265
total quotient ring, 121
trace
absolute trace, 85
of an algebraic number, 128
of Frobenius, 106, 219, 226, 286
over finite field, 84, 89, 324
transitivity of over finite field, 89
transcendental element, 61, 79
transcendental extension, 61
trapdoor one-way function, 10, 406, 425
trial division, 239, 244, 248, 250, 254, 261, 269, 270, 274, 278
triangle inequality, 146, 455
trusted third party, 372
TTP, *see* trusted third party
TWINKLE, 254, 307
TWIRL, 307

UBASIC, 182
UFD, 40, 43, 58, 122, 123, 135, 140, 239, 489
uniform distribution, 162
uniformizer, 94, 114, 141, 288
unique factorization
in a ring, 40

of fractional ideals, 134
of ideals, 131, 133
unique factorization domain, *see* UFD
unit, 33, 42
group of units, 33, 82, 87, 107, 139, 142, 144, 204, 260
unit vector, 66
unitary transformation, 442, 454
unknown key-share attack, 336
UP, 521
upper bound, 19

valuation ring, 286
value
absolute value, 143
of a polynomial function at a point, 93, 119
of a rational function at a point, 94, 113
variance, 160, 162, 163, 164, 166, 409, 414
vector space, 65, 126, 438
complement subspace, 72
finite-dimensional vector space, 65, 67
infinite-dimensional vector space, 66, 67
product of vector space, 65
quotient vector space, 68
subspace, 67
verifier, 340, 366, 370

Weierstrass equation, 98, 108, 111, 218, 289
Weil pairing, 282
Weil-descent algorithm, 308
well-ordered set, 19
well-ordering principle, 19
Wiedemann method, 299
Williams' $p+1$ method, 247, 307
Wilson's theorem, 47, 524
window method of exponentiation, 191
witness, 369, 371
word
addition of words, 182
in computer memory, 174, 181, 205
multiplication of words, 183
subtraction of words, 183

xedni calculus method, 282, 289, 308
XSL attack, 478
XTR, 333, 340, 349
–DSA signature, 349, 365, 378
generation, 351
verification, 351
–Nyberg–Rueppel signature, 365
–Schnorr signature, 365
decryption, 328
encryption, 323, 378
exponentiation in XTR, 326, 349
group, 324
key pair, 327, 350
multiplication in XTR, 350
supergroup, 324

Zech's logarithm table, 87, 210
Zermalo's well-ordering principle, 19
zero, 94, 115, 282
zero-divisor, 33, 42, 121
zero-knowledge protocol, 7, 370
Zorn's lemma, 20, 124, 524